

Turtle Beetle

Juho Pentikäinen (captain), Mikko Pihlanko, Pekka Pihlanko, Juha Helenius, Eero Tuhkanen, Mika Matilainen, Milla Sairanen, Timo Oksanen (instructor), Jari Kostamo (instructor), Petro Tamminen (instructor)

*Aalto University, Department of Automation and Systems Technology,
Aalto University, Department of Engineering Design and Production,
University of Helsinki, Department of Agricultural Sciences*

<http://autsys.tkk.fi/en/FieldRobot2010>

email: jpentika@cc.hut.fi

P.O. Box 11000

FI-00076 AALTO

FINLAND



Abstract

Turtle Beetle is a robot build by the students of Aalto University School of Science and Technology for the Field Robot Event 2010, in Braunschweig, Germany. The robot was designed to navigate between maize rows, identify weeds, handle them effectively and sow seeds in the gaps in the row. This paper describes the building process and used methods for controlling the device. This was the sixth time when Aalto University (the former Helsinki University of Technology) students participated in the contest. In the robot design a clear influence from earlier robots can be seen. However, everything is build and coded from a scratch. The original feature of the new robot design was a pneumatic system, which was used for active suspension, weed destroying and seeding. Also a machine vision as the main navigation method was replaced with a laser scanner. To increase modularity from previous years, most of the electronics was built on a separate plate (plexiglass) that was easy to install on the robot when the chassis was completed. Building the electronics on separate plate made it possible to start the testing of electronic assembly in parallel with mechanical construction.

The planning of the robot was started in September 2009, the actual building started in January 2010 and robot was mechanically and electrically finished in May 2010. The software was developed in test environment in the very same time. Test environment consisted of Matlab simulator and the plexiglass.

The robot was built by the team including design and machining. The completely self-made software was built with Matlab/Simulink, Visual Studio, CodeVisionAVR and LabVIEW -software.

The robot participated in Field Robot Event 2010 in June 2010. In overall competition, the Turtle Beetle robot gained the third prize, and in freestyle competition the second prize.

Keywords: Robot, Maize Field, Navigation, Machine Vision, Suspension System

Contents

Abstract.....	2
Contents	3
1. Introduction.....	5
2. Mechanical structure	7
2.1. Frame.....	7
2.2. Axle Module	8
2.3. Wheels and Tires.....	10
2.4. Agricultural Machinery	11
2.4.1. The seeder.....	11
2.4.2. The sprayer.....	12
2.5. Cover	14
3. Electronics	14
3.1. Computers	14
3.2. Controllers.....	15
3.2.1. Motor drivers.....	15
3.2.2. Sensor interfaces.....	16
3.2.3. Communication.....	16
3.3. Batteries.....	17
3.4. Battery Mode Switch.....	17
3.5. Sensors.....	18
3.5.1. Inclinometer and compass.....	18
3.5.2. Ultrasonic and infrared sensors.....	20
3.5.3. Laser scanner	20
3.5.4. Web camera.....	21
3.6. Electronics modular assembly	21
4. Algorithms and methods.....	22
4.1. Row Detection	22
4.1.1. Detection using only ultrasonic sensors	22
4.1.2. Moving average, ultrasonic and infrared sensors.....	23
4.1.3. Using the laser scanner and pre-defined areas.....	24
4.1.4. Laser for finding maize row.....	25
4.1.5. Recursive least squares	25
4.2. Row Navigation.....	26
4.2.1. Independent PID-controllers for the front and rear end of robot.....	26
4.2.2. Advanced PID-controllers for angle error and position deviation	26
4.2.3. Drive to target point	27
4.3. Row End Detection	27
4.4. Turning Methods.....	28
4.4.1. Simple turning.....	28
4.4.2. Wrong-way turning.....	28
4.4.3. Advanced turning.....	29
4.5. Weed Detection	29
5. Programming techniques and communications	32
5.1. Matlab & Simulink	32
5.2. LabVIEW and NI Vision	33
5.3. Visual Studio.....	36
5.4. Telecommunications.....	36

5.5. Parameters	36
5.5.1. Parameter Slots	37
6. Testing.....	38
6.1. Individual/unit testing	38
6.2. Simulator.....	38
6.3. Test Field	40
7. Discussion	43
8. Conclusions	44
Acknowledgments.....	44
References	45
Appendix 1.....	46
Appendix 2.....	47
Appendix 3.....	49

1. Introduction

Turtle Beetle is a robot built for Field Robot Event 2010 by Aalto University School of Science and Technology students. Aalto University (Former Helsinki University of Technology) students have participated in the event for several years. However, the robot is build from a scratch every year and previous results are not used directly. Most of the ideas for this year's robot can be found in the previous robot "Easywheels" (Kempainen et al, 2009), but for example all the program code redesigned by the students and basically all the machining and electrification is made by the students. However, building a robot and its program code is quite a complicated task and advice and support from the instructors have been essential for completing the task.

The robot's main task is to navigate between the maize rows, identify weeds, handle them effectively and resow seeds to gaps in the row. The robot's main navigation uses a laser scanner, ultrasonic and infrared distance sensors. There are two computers onboard, one for machine vision and the other for the main program. In addition to two computers, up to four microcontrollers are used onboard; two of them drive the motors, the third connects sensors, user interface and controls pneumatics, and the fourth computed inclination.

Also the mechanical system is quite a complicated. The vehicle is a four-wheel-drive system with four wheel steering. Suspension is made especially to keep all the wheels on the surface to avoid slipping and to keep chassis as vertical as possible. The mechanical suspension was improved by adding an active pneumatic roll compensation system.

Most of the differences between the concepts of the previous years and this year are: using a laser scanner for navigation, improved axle module design, improved electrification, semi-active suspension, pneumatically working weed destroying tool, pneumatically working patch seeding tool and improved overall design. Totally new ideas are the use of pneumatics for weed destruction, the seeder and the platform auto balancing system which keeps the robot leveled relative to ground.

Also LabVIEW, which was used for machine vision development, was introduced as a new programming language and tool to the project. The main program was build with C# in Visual Studio compared with C++ in previous year. C# was used in the earlier robot, but this was the first time to use C# together with Windows Embedded CE.

The planning of the robot started in September 2009, actual building started in January 2010 and robot was mechanically and electrically finished in May 2010. The software was developed in test environment in the very same time. Test environment consisted of Matlab simulator and "plexiglass".

Most of the robot was built by the team including design and machining. The completely self-made software was built with Matlab/Simulink, Visual Studio, Codevision and LabVIEW -software.

The main mechanical design goal was to construct the robot from modules. The modular structure contains the frame and suspension, axle modules, agricultural machinery, the platform for electronics, and the body.

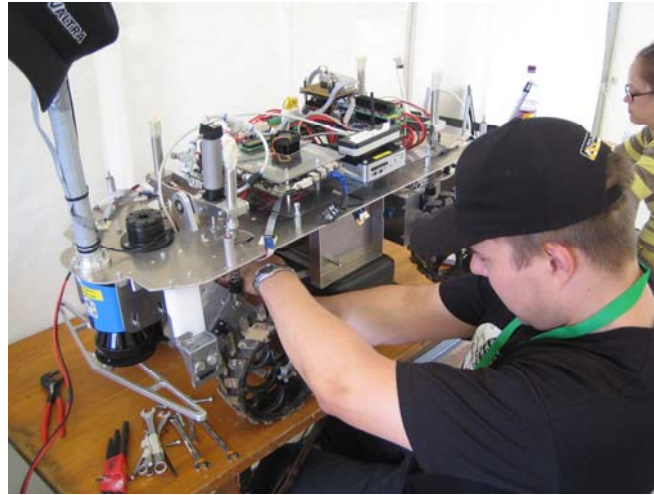


Figure 1: Building up a robot

The modular structure held several benefits. In the design phase the CAD-design of the modules was divided with the team members and also the assembly of the modules could be done independently. The modular structure is very practical in everyday use and servicing the robot, because if a module breaks down it can be easily replaced by a new one or fixed without disassembling the whole robot.

All the metal parts of the robot are made from aluminium, because of its light weight and cost quality ratio. The sheet metal parts were made by laser cutting (Laserte) and the other parts were machined by the team members.

2. Mechanical structure

2.1. Frame

All the mechanical parts were modelled with Solid Edge v20, besides the cover which was designed with Pro/ENGINEER.

The main design aim for suspension was to engineer a pneumatic anti roll bar. There is an air compressor installed inside the frame. The compressor keeps the pressure of the air reservoir in the range of 5...8 bars. The air compressor is controlled with a microcontroller, which regulates the pressure by using a pressure sensors installed in the air reservoir. A speciality of the suspension is that the upper ends of the shock absorbers are connected to moving swings. The front swing angle can be pneumatically adjusted in order to keep the robot's frame in a horizontal position. The front swing can be locked if the roll compensation is not needed. (Figure 2)

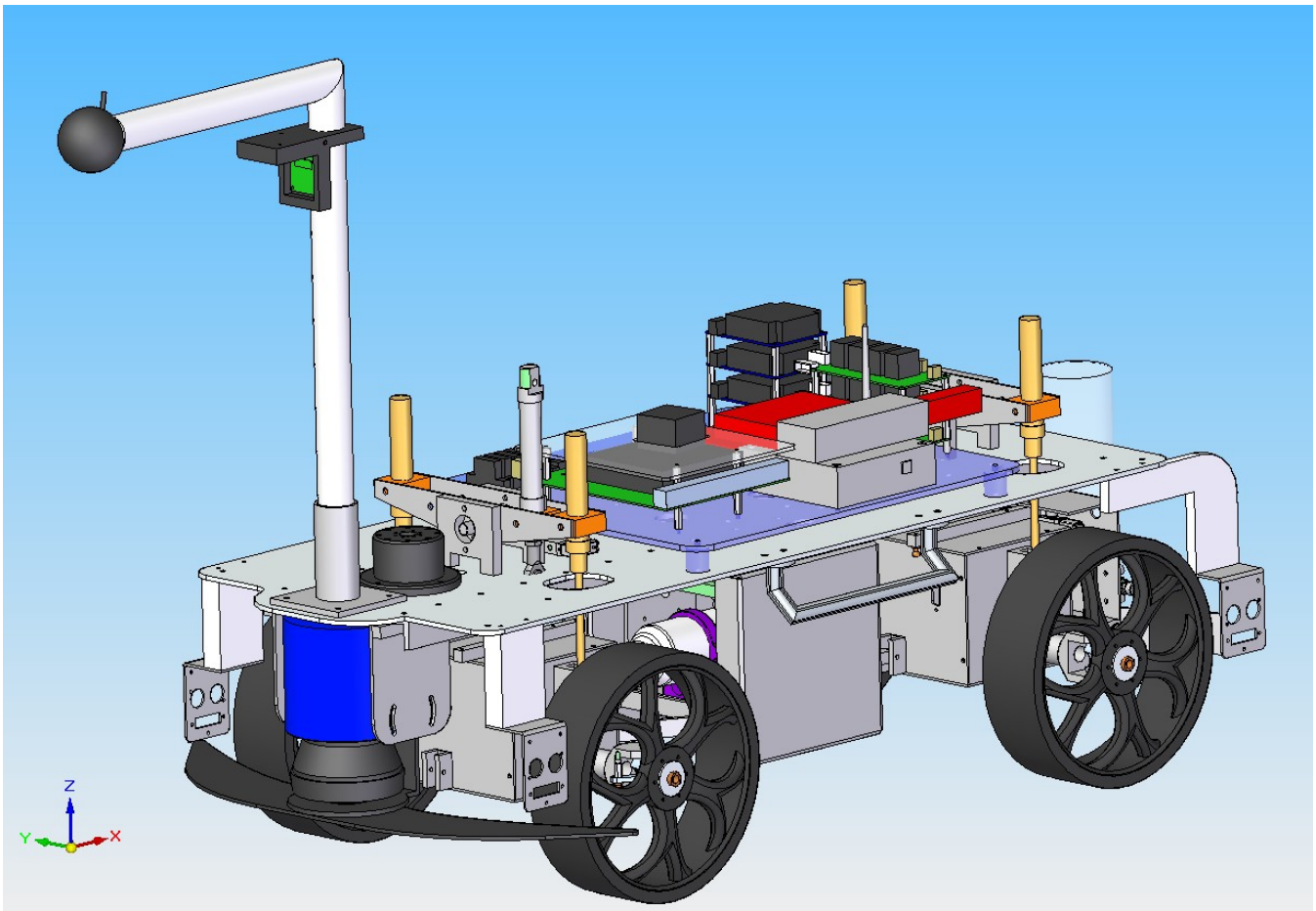


Figure 2: CAD-model of the Turtle Beetle

The pneumatic equipment used in the robot is provided by the FESTO company. In Figure 3 all the pneumatic valves and the suspension absorbers are shown.



Figure 3: Pneumatic valves and suspension absorbers from one of our sponsors FESTO

2.2. Axle Module

In the preceding robot, "EasyWheels", everything required to drive the robot was encapsulated to "axle modules". For EasyWheels three identical axle modules were constructed, one to act as a front axle, the other as a rear axle and the third one was a spare. The idea was to pack all the required components for steering and driving to one module, and this was reported to be such a good solution that the same concept was selected for Turtle Beetle. The modular structure was found convenient for varying the construction of the EasyWheels robot with different tasks. The modular structure also enabled a fast recovery from a malfunction situation, because the whole axle module could be replaced with a spare one in a few minutes. Hence the EasyWheels robot could carry out its tasks on the field without long repair periods. (Kemppainen et al, 2009)

The drive train of the axle module is illustrated in Figure 4. The red ellipse encloses a Tamiya sport tuned DC-motor, which produces the torque for the wheels. Inside the blue ellipse is the planetary gear box (16:1 ratio) and in front of the gear box is a clutch bell (yellow ellipse). Differential is enclosed in a green ellipse and obviously the drive shafts are connected to it although they are not present in Figure 4. The clutch bell, differential and drive shafts come from Xray RC model car.

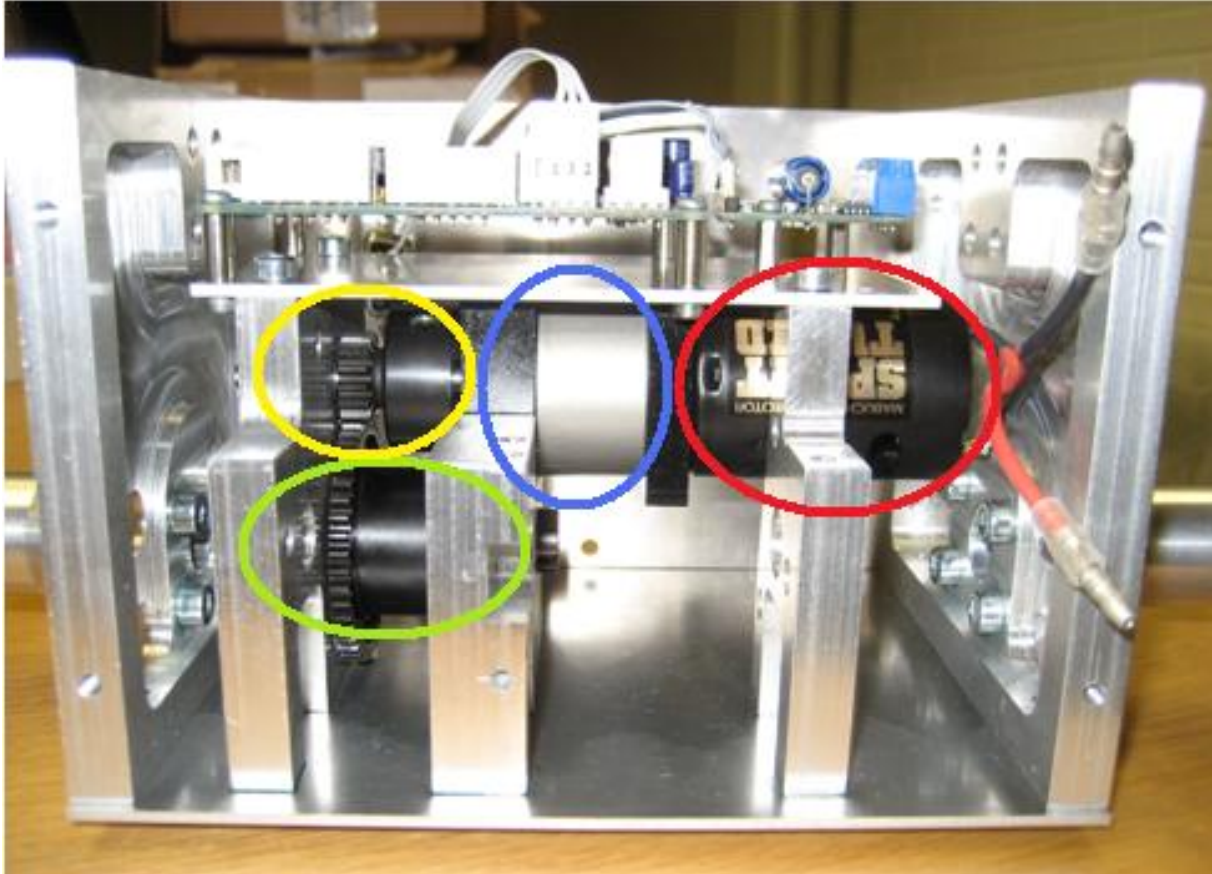


Figure 4: Drive train

The power for the steering system is produced with a Hitec HS-7954SH RC-servo (enclosed by a red rectangle in Figure 5). The steering torque of the RC-servo is delivered to the steering blocks (enclosed by a blue rectangle in Figure 5) via a steel cable. Inside the yellow rectangle are a pulley and a potentiometer, which are attached to the RC-servo's output axis. The steel cable circulates the pulley and the ends of the steel cable are fastened to the steering blocks. The angle of the RC-servo is determined with the potentiometer, which is needed for the PI-control of the steering system. The DC-motor is also PI-controlled and the angular velocity data of the motor is provided for the micro controller by an optical rotary encoder.

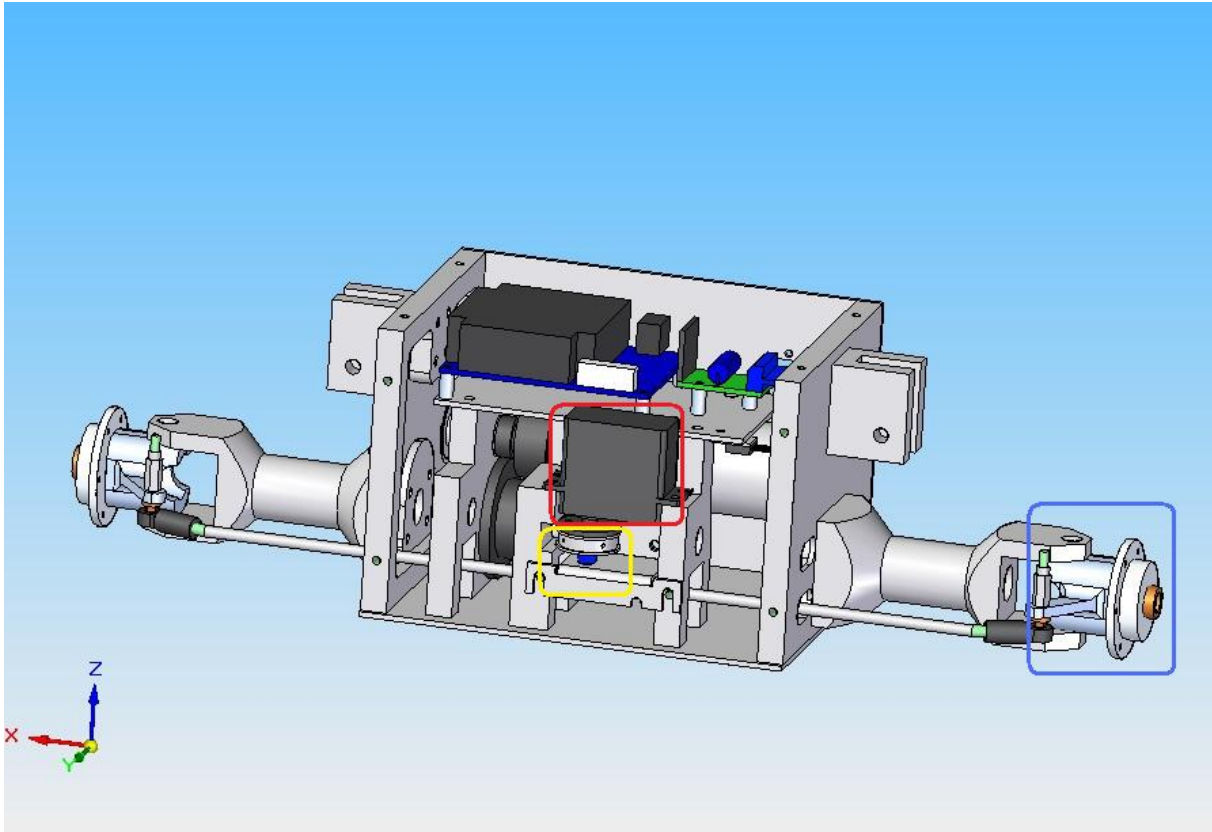


Figure 5: CAD image of axle module components

The objective with the axle module of the Turtle Beetle was to improve the issues that came up with the axle module of EasyWheels. The main improvement goal was to reduce the weight of the axle module and therefore reduce the unwanted non-suspended mass of the whole robot. The previous self-made motor driver was also changed to a much smaller commercial one. The steering blocks of EasyWheels came from a RC-vehicle and they already had a certain amount of camber angle. Thus the steering blocks had to be modified so that the camber angle was zero. Because of this problem, the steering blocks of Turtle Beetle were designed and made by the team members.

2.3. Wheels and Tires

The wheels were designed by the team and carved from a large cylinder of plastic (Figure 6).

The plastic cylinder has cut to eight equal pieces. The work pieces have been carved first inside. Then we put a cylinder made from steel inside to support the fixing for outside carving which has made in two parts. The cylindrical part has carved first. After flipping the piece and putting the support ring the face has carved with the same tool. After turning operations the rim was almost finished but we decided to mill some spool shapes to achieve good-looking and lighter rims. The piece was fastened in the chuck which was installed in the mill table. Then the shapes has milled by using three dimensional tool paths. The surface of the tire has cut from the big RC- truck car tire. Only 60 mm wide strip was needed. Strip has cut with the knife tool. The rubber was fixed on the wooden support block in the manual lathe. Then carefully turning the spindle by hand the cutting was possible. The strips has glued to the rim with the adhesive.



Figure 6: CAD image of axle module components

2.4. Agricultural Machinery

Two agricultural machines were designed for the robot in order to take part in weed destruction and freestyle tasks. The machines can easily be attached to the rear axle module with three bolts. They also have quick couplings for pneumatic and electrical connections. The compressed air comes directly from the robot's pneumatic system and valves and solenoids are driven by three Opto Output amplifiers, which are in turn controlled by a microcontroller on the plexiglass.

2.4.1. The seeder

The seeder was created for the freestyle task of the competition. The idea was that when a gap was detected in a maize row, the robot would stop and sow a seed in place of the missing plant. The idea was similar like in 4M (Backman et al, 2008). The seeds are actually airsoft gun pellets, which are put into vertical pipes (Figure 7, the red ellipse). Solenoids (yellow ellipse) are used to drop the seeds one by one into elastic tubes (not shown in Figure 7) which are connected to longer pipes (blue ellipse). When the seed finally drops into the longer pipe, the pneumatic cylinder (green ellipse) pushes the pipe downwards and the seed is blown out by compressed air. The seeder has three pneumatic valves (black ellipse): two 5/2 valves to control the cylinders and one 2x3/2 normally closed valve to control airflow into the long pipes.

It was expected that the weakest link of the seeder might be the solenoid system designed to drop the seeds and actual tests proved this to be true. The solenoids were not powerful enough to compress return springs that hold slide closed and the seeds won't drop. This problem can be corrected with more powerful solenoids. On the other hand, the pneumatics work well which is the most important thing in the freestyle task because moving cylinders and whistling valves give the judge and audience a feeling that something cool is happening.

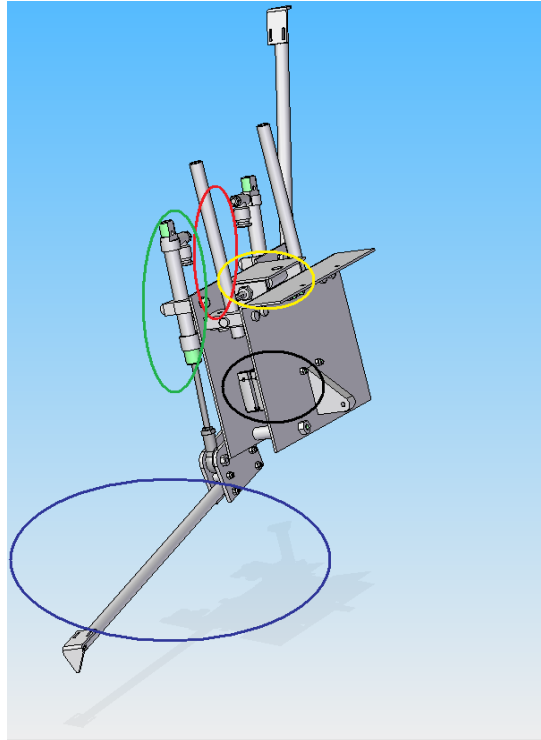


Figure 7: The seeder

2.4.2. The sprayer

The sprayer is to be the weeds' worst nightmare. When the robot detects a weed in the maize field with machine vision, it stops and sprays herbicide (water) to kill the weed. The operating principle of the sprayer machine is very simple. The bottle contains water and is connected to two vacuum generators (Figure 8, blue ellipse) with pneumatic tubes. One 2x3/2 normally closed pneumatic valve (red ellipse) is used to control airflow to the vacuum generators. When the valve is opened, vacuum sucks water from the bottle to the vacuum generator where it is mixed with air and finally sprayed to the side of the robot. The vacuum generator is not meant to be used for this purpose, but seems to work nicely. The amount of water can be adjusted by flow control valves which are shown in Figure 9.

Thanks to its simple structure, the sprayer performed well in tests. The only problem was that water flew spontaneously out through vacuum generators due to gravity. Fortunately, we were able to restrict the leak by adjusting the flow control valves.

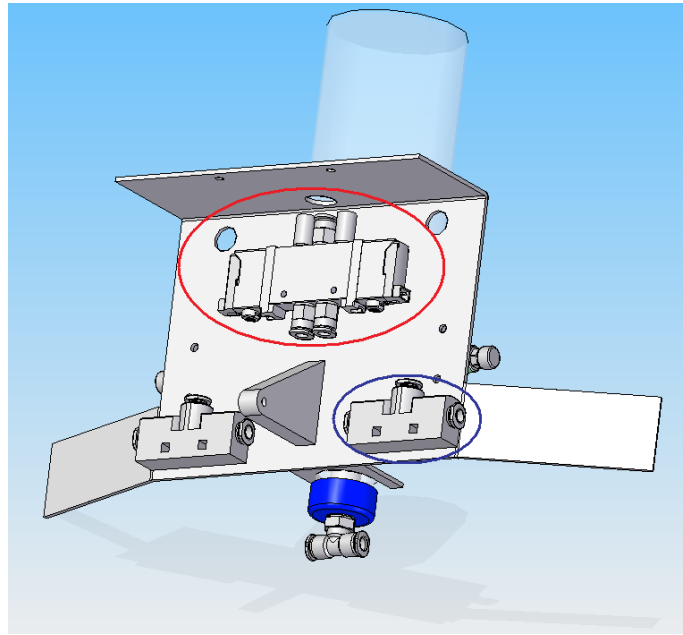


Figure 8: CAD image of the sprayer

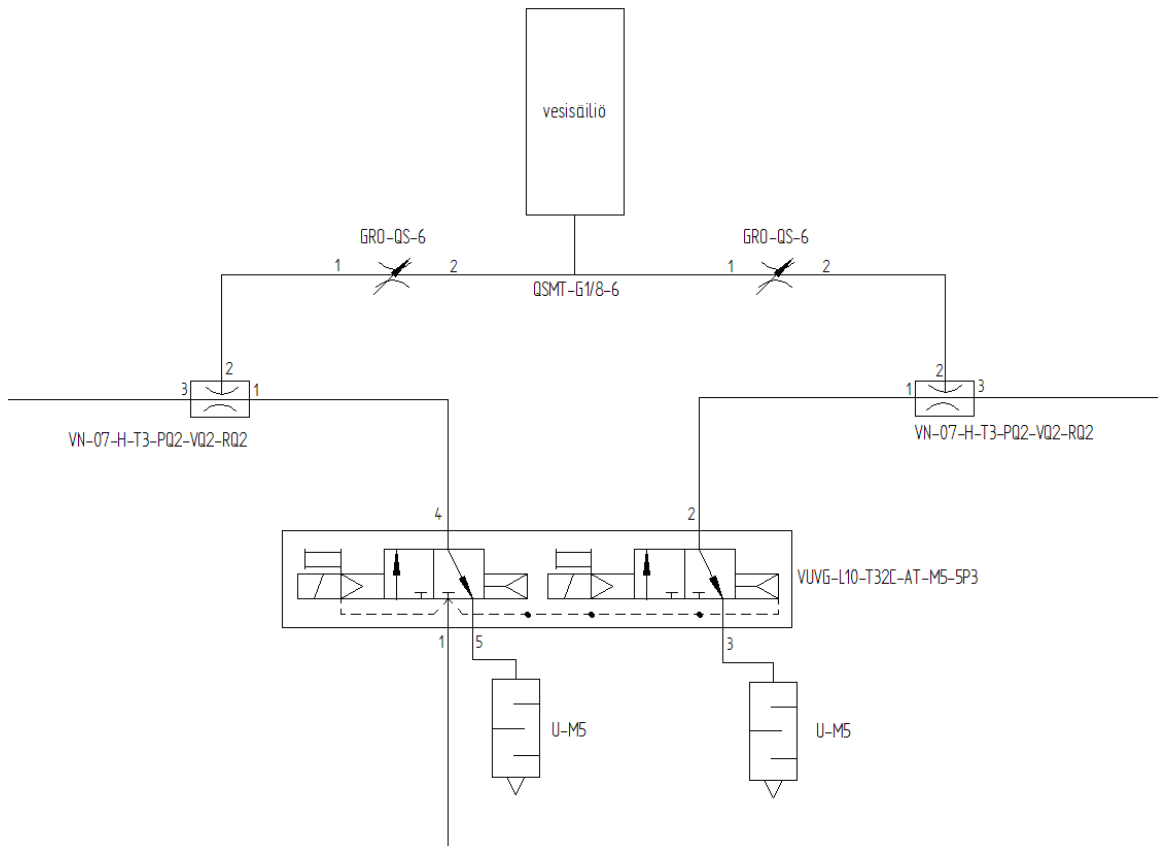


Figure 9: The sprayer's pneumatic diagram

2.5. Cover

The cover for preceding robot "EasyWheels" was designed and manufactured by a sponsor, and this was not the fact in case of Turtle Beetle. The idea to have nice looking, single piece cover was decided. Solid Edge was used use design all the robot mechanical parts etc. but the software does not suit for designing smooth surfaces, like Turtle Beetle has. Fortunately, an industrial designer Pekka Kumpula from S.E.O.S design Ltd, helped us by designing the cover outlook with Pro/ENGINEER. Industrial designer had an idea that usually robots have many bugs inside, so the design was to be a Big Bug (Beetle is one of many species of bugs).

The shape of the body has made with vacuum forming, which is a process whereby a sheet of plastic is heated to a forming temperature, stretched onto a single-surface mold, and held against the mold by applying vacuum between the mold surface and the sheet. The mold has milled to the SIKA polyurethane tooling board. The mold was milled by a team member, but the team got some help for vacuum forming from a local workshop. The sheet was 3mm thick VIVAK (Polyethylene terephthalate). Formed sheet has trimmed and the body has painted inside to make a shiny finish.

3. Electronics

3.1. Computers

The robot has two computers onboard (Figure 10). ICOP eBox-4300 minicomputer with 500 MHz VIA Eden processor and 512 MB DDR2 is for the main program ("the brains") and Toradex Woodpecker with 1.6 GHz Atom processor (Z530) and 1024 MB of DDR2 is only for weed detection ("machine vision").

The main program loop runs in eBox which is attached to three microcontrollers in total with serial ports and to a WLAN router with RJ45 cable. One of the serial ports is made with USB-serial converter. Router is used for communicating between eBox and Toradex Woodpecker.

The router is also used to connect a laptop to eBox or Toradex Woodpecker. This can be used for the remote desktop connection or the remote user interface (Remote UI) which was made for the testing and adjustment of the robot. Also updating the software to eBox goes over WLAN. However, the robot can move independently without any laptop. Communication via router was built with UDP-protocol for simplification reasons.

Woodpecker uses 12V voltage which is the same as used the robot's electrical system. However, eBox uses 5V voltage like the WLAN router so there is a DC/DC converter onboard (Texas Instruments TPS5430 in evaluation board).

eBox has Windows Embedded CE 6.0 installed as operating system and Toradex Woodpecker has a regular Windows XP. Both worked well for the competition and for learning purposes. eBox should have had somewhat more computing power, as it limited developing the algorithms partially. Perhaps structure and coding would have been more simplifier if both computers would have used the same operating system for example Windows XP Embedded.

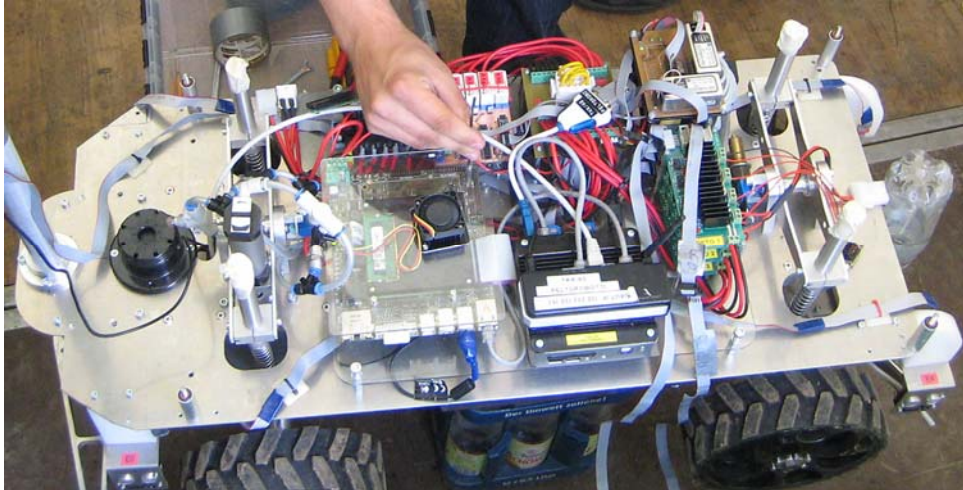


Figure 10: Robots main electronics

3.2. Controllers

Turtle Beetle has four Atmel ATmega128 microcontrollers on Futurlec ATmega proto boards (Figure 11), which were used also in 4M (Backman et al, 2008) and in EasyWheels (Kemppainen et al, 2009) and were reported being good. Front and rear axle modules have their own controllers for motor and servo control and there are also two controllers on the plexiglas. One of them calculates robot's inclination angles (roll and pitch) from the information about inclinometers and gyroscopes, the other is used to read sensors (ultrasonic and infrared range finders, compasses), to control pneumatics and to read and write local user interface I/O. Control signals for pneumatic valves, a buzzer and a compressor relay are amplified in three Opto Output amplifiers. User interface includes LCD display and eight LEDs and buttons, which are controlled by two Microchip MCP23S08 GPIO expanders in SPI bus.

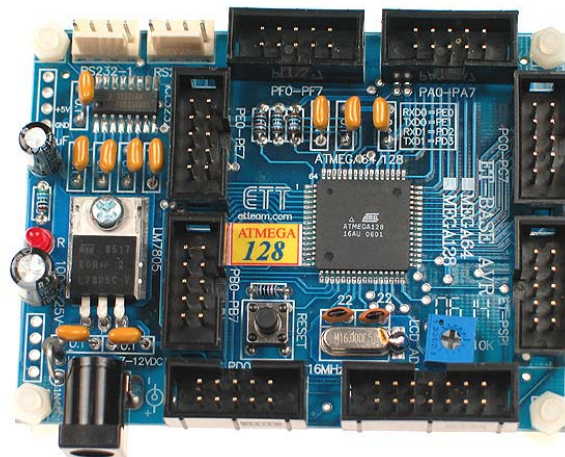


Figure 11: Futurlec ATmega proto board

3.2.1. Motor drivers

The motor is driven with PWM signal which is amplified in Pololu 18v15 motor driver (Figure 12). For the motor driver two inputs are required: the direction as a bit, and the PWM signal to be amplified. Controller

measures speed with the Avago Technologies optical rotary encoder and uses the feedback to keep the desired speed if RPM value is set. The PWM output is then obtained by a PI controller. The servo position can also be set by PWM or desired angle. Angle feedback is given by potentiometer that is connected directly to servo's axle.

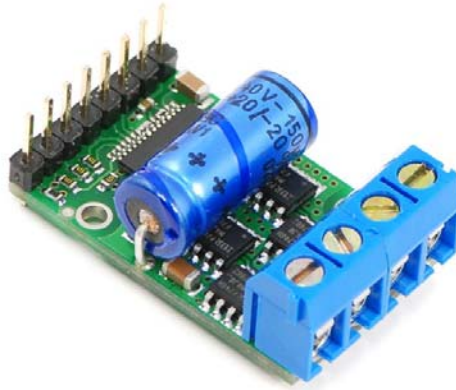


Figure 12: Pololu 18v15 motor driver

The very first field tests revealed a very serious problem with the motor drivers. The big blue capacitor (in Figure 12) easily overheated and blew up under normal driving conditions. The robot had then to be stopped immediately to avoid more damage. This happened a few times but we couldn't find any good reason, since the motor drivers were used according to manufacturer's instructions and specs. The solution was to replace the original capacitor with more heat resistant tantalum one and reduce the PWM frequency remarkably (from 16 kHz to 2 kHz), which prevented the heat from rising too high. Besides a tantalum capacitor, a much bigger electrolyte capacitor was installed in parallel. These actions significantly improved reliability, but actually we were able to blow up also one tantalum capacitor in the event test field, just a day before competition.

3.2.2. Sensor interfaces

Infrared range finders' output is analog voltage (0-5V) which is read by the controller and converted to distance by eBox. Communication with SRF08 ultrasonic range finders and CMPS03 compasses is via I²C bus. Each sensor in I²C bus has a unique address to separate it from other devices. The controller is the bus master that commands sensors and reads measurement data; sensors are slave devices. The I²C bus is relatively easy to use, but it seemed to be a bit unrobust. For example, if one device stopped working because of a broken cable, the whole bus died.

3.2.3. Communication

Communication between controllers and eBox is via RS-232 serial bus (19200 bps, 8N1). A special protocol was developed for the communication. Data is always sent in one frame, which starts with 0xFF followed by a unique device ID. The next bytes are actual data. After data bytes comes a checksum byte, which is calculated from data bytes and used to prevent errors in communication. Controllers and eBox both calculate the checksum for each message and if there is a mismatch, the whole message is rejected. The frame ends with bytes 0xFF, 0xEE. All 0xFF bytes in the data are replaced with two bytes (0xFF, 0x00). All the measurement data is sent to eBox for further analysis; eBox sends back parameter and set point values, like PI controller parameters or desired speed.

3.3. Batteries

The main idea with the batteries was to use equal size batteries which are easily changeable. In addition there was a goal to make the robot lighter than earlier years so it was decided to use lithium polymer batteries (LI-PO). These batteries are quite light compared with lead acid batteries, but there are some differences. One of the most important thing is that battery voltage must be between 3.0 and 4.2 volts/cell. So it is needed to use a cutoff circuit to prevent discharging the battery voltage too low. The charger must also be "intelligent". The charger starts charging with constant current set by user and when the voltage reaches 4.2 volts per cell it starts charging with constant voltage. Then batteries are about 95-90% full. The last 10-15% takes time about the 1/3 of the full load charging time so it is often useful to end charging if time matters. LIPOs are also very vulnerable; there is only a very thin layer of plastic film over the battery core. Misuse of batteries can cause a fire.

Turtle Beetle uses four batteries (3 Cell 5000mAh, Figure 13) placed in the cases (two packs in each case). In total two sets of batteries (8 pieces) were bought, so it is possible to charge one set while running with the others. So it has the energy to make a one-hour continuous trip on the field (or to make a huge LIPO fire). The nominal voltage of each battery was 11.1V while the legal usage voltage is from 9.0 to 12.6V.



Figure 13: Batteryback

3.4. Battery Mode Switch

The electrical devices of the robot were divided into different power circuits. The purpose of this was that we could get different battery switch modes which are Racing mode, Debug mode, and External power supply mode

The battery switching can be done with a single circuit board, which has input to batteries and outputs for the power circuits. The circuit board holds a 15-pole socket in the middle (Figure 14). All four batteries and the power circuits have their own connectors in this 15-pole socket. By changing a differently wired plug to this socket we can assign the batteries to different power circuits.

In racing mode the front - and the rear axle modules have their own batteries for DC-motor and RC-servo power supply. In debug mode a single battery is supplying power for both axle motors and therefore more

power can be supplied for the computers and micro controllers. In the external power supply mode we can use external power supply for running all of the electrical equipments.

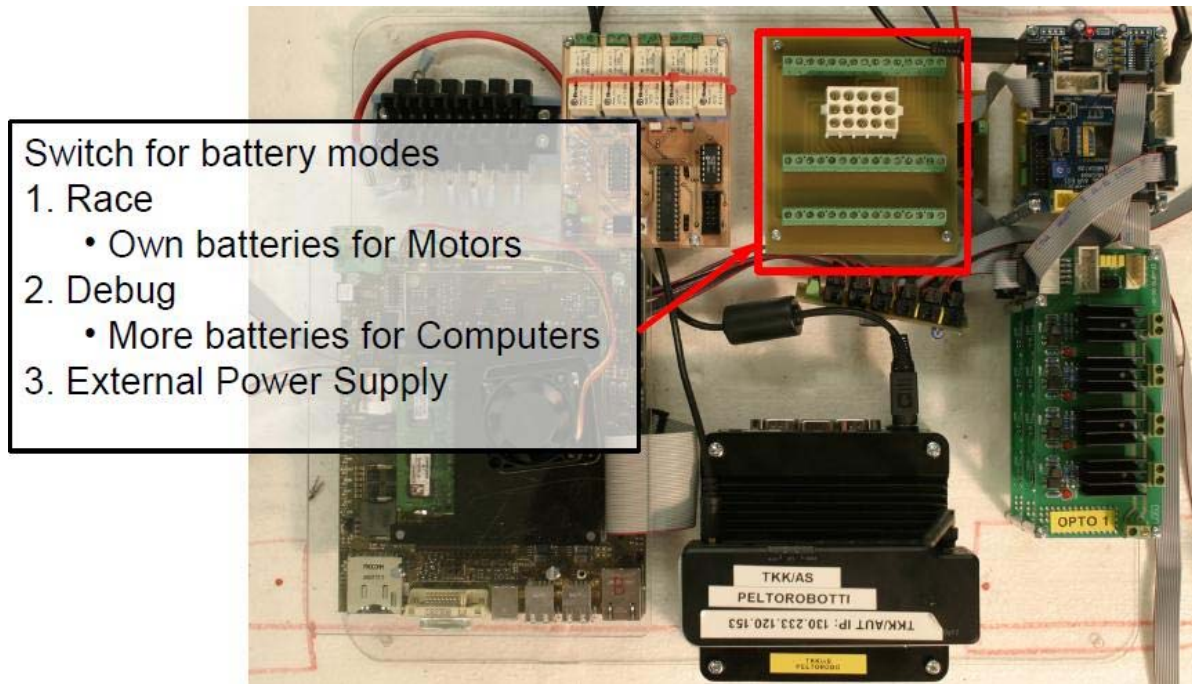


Figure 14: Battery switch modes

3.5. Sensors

The robot has four infrared sensors and four ultrasound sensors, a SICK laser scanner, two compasses installed at 90 degrees angle relative to each other for 3D information, combined gyroscope and 3-axis accelerometer, and an ordinary web camera for weed detection.

3.5.1. Inclinator and compass

The robot has two 2D electronical compasses (CMPS03) and two gyroscope-accelerometers combos (VTI SC-1120) for taking the accurate measurements of the direction (Figure 15). The idea is that gyroscopes and accelerometers are used as an inclinometer, which gives the roll and pitch of the robot. Compasses are very sensitive to tilt because the measured component of the Earth's magnetic field is measured in the wrong angle. For this reason it was necessary to compensate the compass error with inclination.



Figure 15: Compass and gyroscope

The compass had to be calibrated carefully. Together we had four compass measurements which were components of Earth's magnetic field. The compass angle could be calculated directly as a tangential angle of the components. However, this measurement could lead to very odd compass angles without calibration, because measurements were not located around origin as presented in Figure 16. The easiest way for calibration was to roll the robot on the floor and draw measurements as Matlab graph. This had to be done in two steps to get the errors of the measurements in x, y and z direction. In the first step, the robot was standing on its wheels while turning it 360°. In the second step, the robot had to lie on its other side while turning it again 360°.

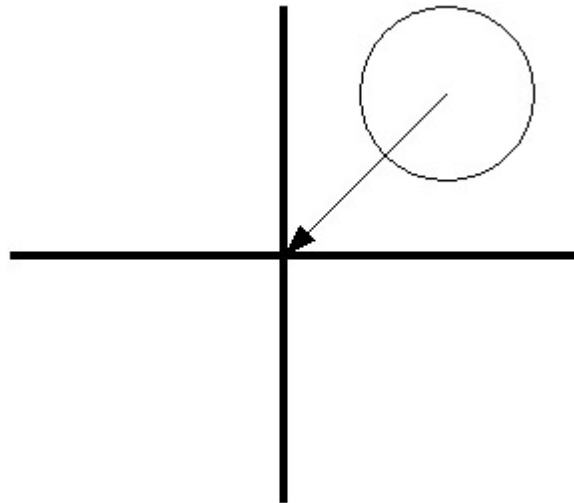


Figure 16: Compass calibration

3.5.2. Ultrasonic and infrared sensors

SRF08 ultrasonic range finders are located in each corner of the robot and measure distance from maize plants. The finders' maximum measuring distance was set to 0.60 m which is enough for operating in a maize field. Longer distance would give echoes from farther rows and from ground. Ultrasonic range finders are fired in every 50 ms and there is a 24 ms phase difference between front and rear sensors in order to prevent false echoes. Front sensors are fired and read before rear sensors. The SRF08 sensors give distance directly in centimeters.

In addition, the robot has four SHARP GP2D12 infrared range finders located below each SRF08 (Figure 17). These sensors give an analog voltage output which can be converted to distance. The maximum practical measuring distance is about 30 cm.

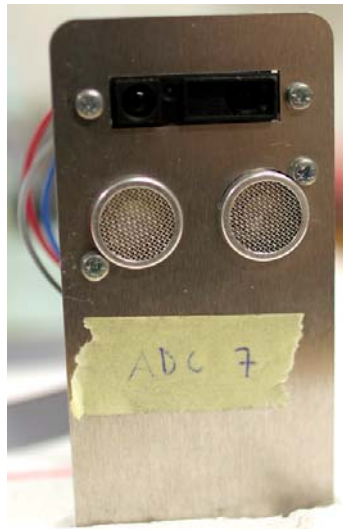


Figure 17: Ultrasonic and infrared module (upside-down)

3.5.3. Laser scanner

Sick LMS100-1000 laser scanner (Figure 18) is located in front of the robot. It is attached upside-down to make sure that laser beam will hit small plants. The scan angle of LMS100-1000 is 270° but only 200° was used because of the geometry of the robot. Updating frequency was set to 50Hz and angular resolution to 0.5°. The scanner is connected directly to the router and communication is based on TCP protocol.



Figure 18: Sick LMS100-1000
laser scanner

The manufacturer has announced that distance measuring range is from 0,5m to 20m but we noticed that it is possible to measure distances as short as few centimeters. Maximum measuring range used was 2 meters.

3.5.4. Web camera

For the weed detection, a mid price web camera was used. It had been proven successful with previous robots (Backman, J. et al.), was relatively easy to obtain and use, and did not cost much. A web camera uses USB 2.0 for data transfer. Theoretically, it is faster than Firewire but in practice slower (FireWire). However, for this application a web camera using USB 2.0 was good enough as the main bottleneck was the processing power of the computer.

The chosen web camera was Logitech QuickCam Communicate Deluxe (product number 961465-0509) by Logitech Inc. It claimed to use a CCD image sensor but we were not so sure about it. Another important characteristic was the “extra wide” field of view, which was about 60 degrees. The wide field of view means that the camera can be positioned lower than otherwise. Logitech QuickCam Communicate Deluxe also uses software to automatically adjust to lighting conditions although that property was not tested for nor intentionally used.

3.6. Electronics modular assembly

Previously, electronics have been firmly attached to the robots body (Backman, J. et al. and Kemppainen, T. et al.). The problem with this structure is that robots body needs to be ready before electronics can be made. That is against of the idea of modular structure.

In Turtle Beetle, all the main electronic components have been attached to a piece of plexiglass. This way it was possible to build up a module for electronics and it was done simultaneous with building up the robot. The ready module can be attached to the robot very fast.

There was also second reason to build a module for electronics. With the module it was possible to start to use the electrical system although the mechanical structure was under construction. This was very important for testing electronics assembly, testing the protocols and software.

4. Algorithms and methods

The robot's main algorithms and methods comprise of sensing where the robot is (row detection), moving the robot (row navigation, end of rows turning) and detecting the weeds (machine vision).

One or more methods for each purpose were developed. All of these methods are described in more details in the following Chapters 4.1 -4.4. The main idea was to at first develop simple and easy to use methods. Later, more sophisticated methods were developed, but the both input and output interfaces of the algorithms had to remain the same, in order to provide easy changing from one method to the other. By doing this way, it was easier to find out what the method should do. It was learning by doing. We could also be sure that in every case we have at least one method that will do its work.

4.1. Row Detection

A few different methods for detecting the position of the robot were developed. All the methods use the same interface.

Inputs to the methods are measurements from the ultrasonic and infrared sensors and the laser scanner.

Outputs are the angle between robot's moving direction and the real direction of a maize row (in degrees, Figure 19), deviation from the center line of the row (in meters, Figure 20), probability that the robot is in the row and quality of row detection.

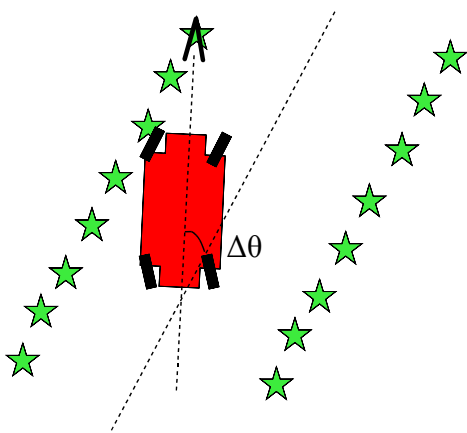


Figure 19: Angle error

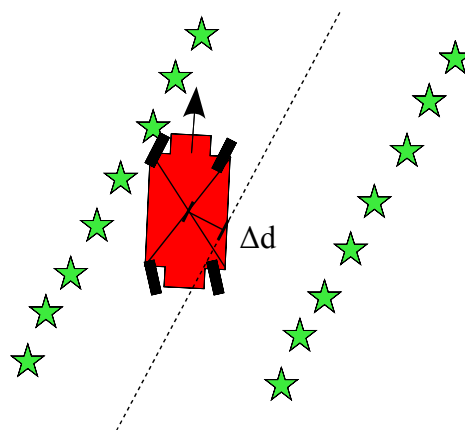


Figure 20: Distance from the center line

4.1.1. Detection using only ultrasonic sensors

The simplest method uses data only from the ultrasonic sensors. If some (or all) of the sensors give invalid measurement, the data will be rejected. The smallest and biggest acceptable measurements can be set simply by changing the value of the parameter.

Rejected measurement will be replaced by the data that is calculated from the last valid measurement of the current sensor and measurement of other sensor as an average. The first option is to use measurement from the same end but different side. The second option is the measurement from the other end and the same side.

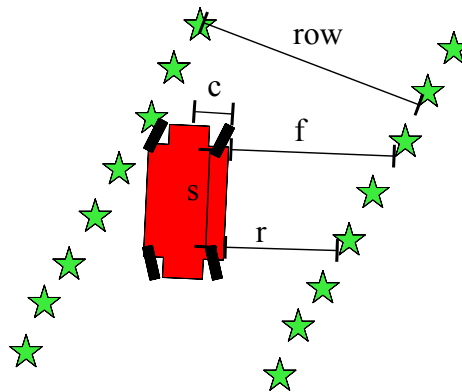


Figure 21: Calculating deviations

Angle error can be calculated by using simple geometry, presented in Figure 21. Symbols f and r represent values that are obtained by using method described above.

$$\Delta\theta = \arctan \frac{f - r}{s} \quad (1)$$

Because sensors are symmetrically around the center point, deviation from center line can be calculated by using average of front and rear sensor readings

$$\Delta d = \frac{f + r}{2} + c * \cos(\Delta\theta) - \frac{row}{2} \quad (2)$$

On field tests it was noticed that this method works quite well. The steering angle tends to oscillate a bit.

4.1.2. Moving average, ultrasonic and infrared sensors

This method calculates the position of the robot first by using data from ultrasonic sensors and then from the infrared sensors. The calculations are made by using the same function for both sensor types. This function uses a few last valid measurements and calculates the (moving) average. If a measurement is not valid, then algorithm uses previous measurements but gives them less weight.

From the filtered values, the angle and distance deviations are obtained by using the same equations (1 and 2) than in the previous method.

Final values for deviations are obtained by calculating weighted average from the deviations of both ultrasonic and infrared sensors.

On field tests it was noticed that this method works quite well. The overall result is quite similar to the previous method.

4.1.3. Using the laser scanner and pre-defined areas

Data from ultrasonic and infrared sensors is used similarly as in the previous method. For laser scanner data, four areas (“boxes”) around the robot have been defined. The average is calculated from the position of plants that fit inside the box (Figure 22). This way the method will get four points from the laser data. Each calculated point will get weight-value according to the number of plants found inside the box.

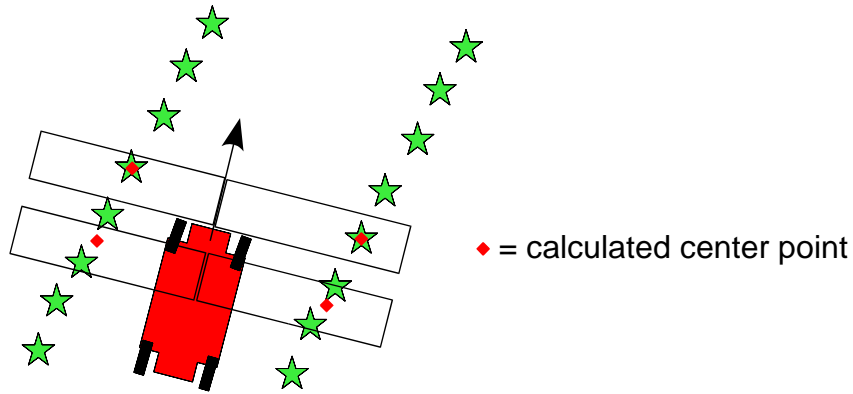


Figure 22: Pre-defined boxes and calculated center points

The next step is that measurements are shifted next to center line and then front end (and rear end) values are combined together using weighted average (Figure 23).

$$point = \frac{l * weightL + r * weightR}{weightL + weightR} \quad (3)$$

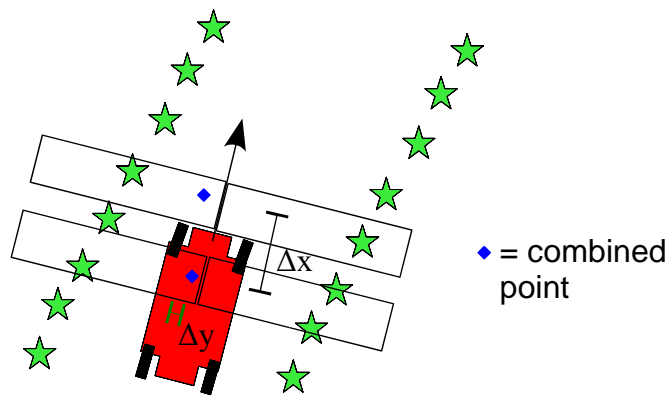


Figure 23: Combined points

Deviations can be calculated quite similar to previous methods.

Angle error

$$\Delta\theta = \arctan \frac{\Delta y}{\Delta x} \quad (4)$$

where

Δx is distance between points in x direction and

Δy is distance between points in y direction

Deviation from center line

$$\Delta d = y_1 + \frac{\Delta y}{\Delta x} * x_1 \quad (5)$$

where

x_1 is the x coordinate of front point and

y_1 is the y coordinate of front point

4.1.4. Laser for finding maize row

This method is experimental and it hasn't been completed. The idea of this method is

- if the measurement given by laser scanner is larger than threshold value, there is a gap
- laser scanner finds small gaps from both side of robot and the assumption is that there is equal amount of gaps on both sides
- the biggest "gap" is the row itself

Now it is easy to select the measurement that represents the center point of a row because it is the middle-most value. The angle of middle-most value is used as angle deviation.

Distance deviation is obtained by using ultrasonic and infrared sensor and methods described in Chapter 4.1.2.

4.1.5. Recursive least squares

The recursive least squares method uses all the data from the laser scanner (if the measurements aren't taken too far from the robot). This method fits the line to data by using recursive least squares algorithm. It is based on a function described in reference (Hyötyniemi H.). The current version is in Appendix 1.

We assume that distance between maize rows is constant that is why all the plants can be shifted to a single "mathematical" row by using modulus. It is possible to use this method only if the angle between the robot and rows is small. To be able to use this method in the general case, the data from the laser is rotated in small steps and each time least-squares is calculated. Finally, the rotation which gives the best correlation is selected. Simulated example figures are shown in Appendix 2.

Summing up angles given by least-squares and rotation angle, the angle deviation is obtained.

$$\Delta\theta = \arctan(k) + rotationAngle \quad (6)$$

where

k is the slope given by least squares method

$rotationAngle$ is the rotation angle which gives the best correlation.

The distance deviation is obtained

$$\Delta d = \frac{b}{\cos(rotationAngle)} \quad (7)$$

where

b is the intercept given by least squares.

While testing this method on the on-board computer, we noticed that it takes too much CPU time. That is why it was decided not to use it on the field test.

4.2. Row Navigation

A few methods were developed for navigation in row. All the methods use the same interface. Inputs to the methods are the output values of the row detection method (the angle between the moving direction and the real direction, deviation from the center line of the row, probability that the robot is in the row and quality of the data). Outputs are the controls for driving and steering motors.

4.2.1. Independent PID-controllers for the front and rear end of robot

This method calculates the distance between the corners of the robot's frame and the maize row. Basically, this is reverse operation to defining angle and distance deviations in Chapter 4.1.1 because optimal ultrasonic sensors would give desired values. Now by using deviations, calculated distances are "filtered".

Left-side distance is given as a set-point to PID-controller and right-side as measurement because by doing this it does not need to use zero as a set-point. Front and rear axle steering are controlled independently.

On field tests it was noticed that this method works well.

4.2.2. Advanced PID-controllers for angle error and position deviation

In this method PID-controllers correct the angle error and the deviation in robot's position. At first it is calculated how far the center point of the front end (and rear end) is from the center line of the row.

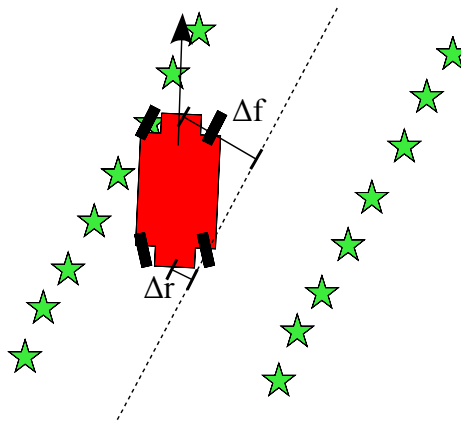


Figure 24: Front and rear end deviations

One PID-controller tries to correct the deviation of the front end by steering front wheels and the other controller the deviation of the rear end by steering rear wheels. The angle error is corrected by subtracting it from controllers' outputs. The result is that in situation presented in Figure 24 angle error makes front wheels turn to the right even more and rear wheels turning less to the right.

On field tests it was noticed that this method works quite well but the result was not as good as independent PID-controllers for the front and rear end of the robot.

4.2.3. Drive to target point

In this method a target point for the robot is calculated. Only front wheels are steering. This method is an experimental method and was not thoroughly tested. Method seems to work in some conditions, but probably a more sophisticated control would be better. Adding for example I and D terms with some limitations could be the right solution.

The idea is that there are two parts in the turning angle of front wheels. The first part is the angular deviation of the robot $\Delta\theta$. The second part is calculated from the distance deviation of the front end Δf and from one tunable parameter dX (Figure 25)

$$\Phi = -\Delta\theta - \arctan \frac{\Delta f}{dX} \quad (8)$$

where

- Φ is the turning angle of front wheels,
- $\Delta\theta$ is the angular deviation of the robot,
- Δf is the distance deviation of the front end and
- dX is the tunable parameter.

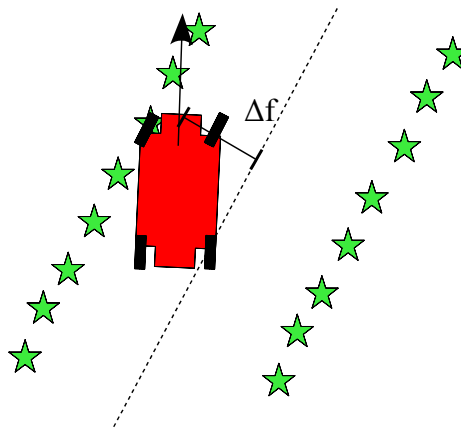


Figure 25: Driving when only front wheels are steering

4.3. Row End Detection

The main sensor for detecting the end of the row is a laser scanner. If the laser scanner does not find enough plants (more than 50) near the robot (closer than 0.8m), it is assumed that the row has ended (Figure 26). Back-up system uses ultrasonic sensors. If none of the sensors detect plants, inside a certain range (from 0.01m to 0.35m), during a certain time (nine program cycles), it is also assumed that the row has ended.

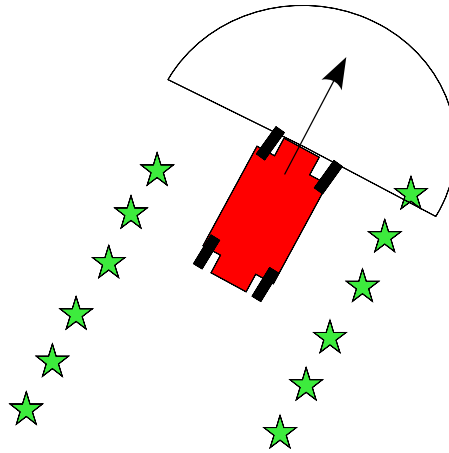


Figure 26: Row end detection by using laser scanner

On field tests it was noticed that this method works very well and normally ultrasonic sensors are not needed.

4.4. Turning Methods

A turning method is activated after the row end has been detected. All the turning logic methods have been implemented by using Simulink Stateflow tool.

4.4.1. Simple turning

The simple turning method uses only angle data from electronic compass and distance data from axle modules. At first robot makes a 90-degree turning to the desired direction. After the turn, the robot drives backwards or forwards depending on the position of the next row. The next step is another 90-degree angle turn. Finally, the robot drives forward until it detects the row. The Stateflow model is shown in Appendix 3.

On field tests it was noticed that this method works quite well. However, it is important to calibrate compass and turning parameters on site.

4.4.2. Wrong-way turning

Wrong-way turning method is very similar to the simple turning. The only difference is that instead of a 90 degree angle turn at start-up, a 270-degree angle turn is done. Because of that it seems that the robot starts to turn to the wrong way.

The idea for this method is that it makes possible to count all the rows while driving in the headland. Counting the rows is important to make sure that we turn back to right row. If rows are not counted, turning is made solely based on odometry.

This method has been tested only in the simulator. According to simulation it needs too much space at the end of the row so it was decided not to use it.

4.4.3. Advanced turning

The basis of advanced turning method is similar to the simple turning method. The advanced feature is that after the first 90 degree turn this method uses data from the laser scanner to keep the distance to the row ends at desired value. It is based on counting the number of plants inside the certain area that is shown in Figure 27.

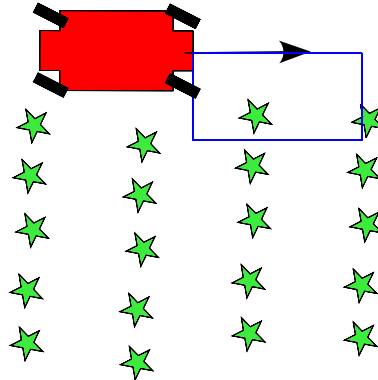


Figure 27: Keeping constant distance to row ends

Two different methods for actual steering were developed.

The first method is very simple.

- if there are not enough plants inside the box, turn both wheels to the maximum angle towards the plants
- if there is a correct amount of plants inside the box, keep wheels straight
- if there are too many plants inside the box, turn both wheels to the maximum angle away from the plants

Second method is more sophisticated and it uses P-controller to steer wheels. There is only one common controller for both front and rear wheels.

$$\Phi = K_p * err \quad (9)$$

where

Φ is the turning angle of wheels,

K_p is gain and

err is the error between set-point and the number of plants inside the box.

On field tests it was noticed that both methods work quite well. Because there were only a few plants inside a box, even a small variation in the number of plants will cause quite a big change in the steering angle while using P-controller. That is way both methods work quite similarly.

4.5. Weed Detection

The web camera is directed at a right angle to the ground so there is no need to geometrically correct the image. After the image acquisition, the luminance or color plane of the RGB image is extracted to get a gray level image. Because the objective is to look for bright white objects on a bright green background, the best option seems to be either red, blue or luminance plane extraction depending on the overall circumstances. The image look-up table needs to be reversed so that bright objects become dark, and the image data is then manually thresholded to a binary image. The threshold value is chosen close to zero. (Figure 28)

The result from the previous stage is a binary image with blobs of varying sizes and forms. To get the daisies, which are nearly uniform in size and shape being small circles, the blobs that are either too big or small are needed to be filtered out, and then from the remaining ones look for circular blobs. (Figure 29)

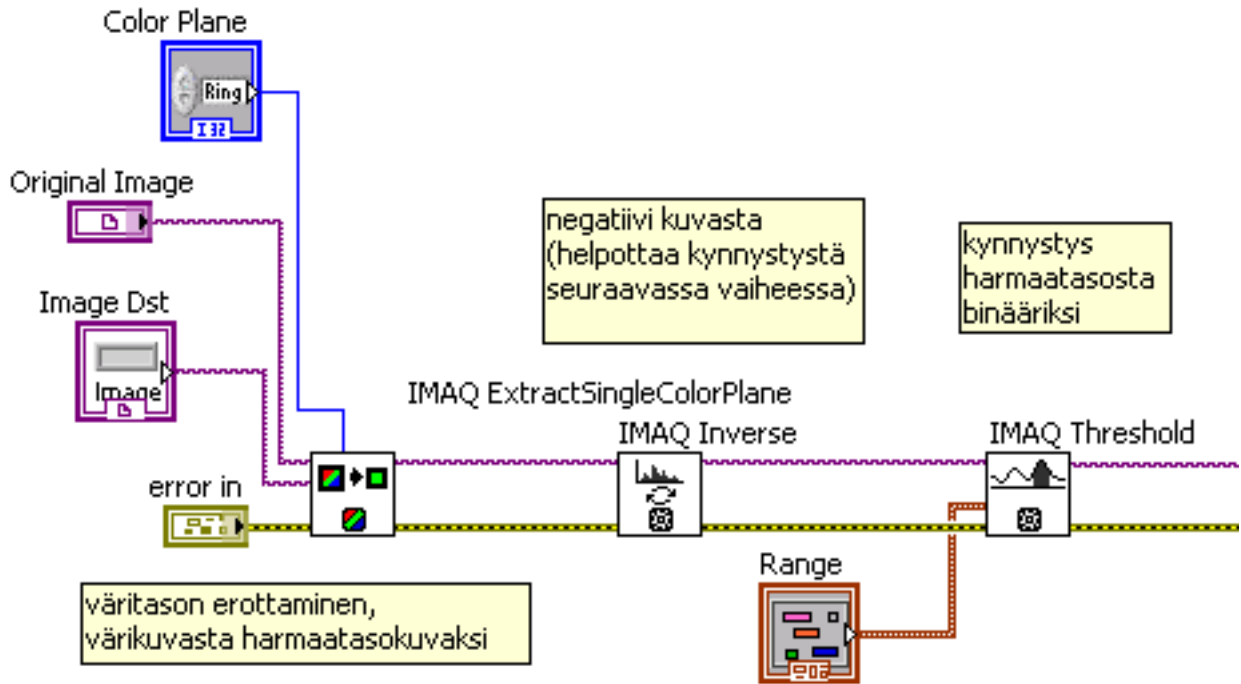


Figure 28: Pattern recognition for weed detection, first half. Color plane extraction, look-up table reverse and thresholding

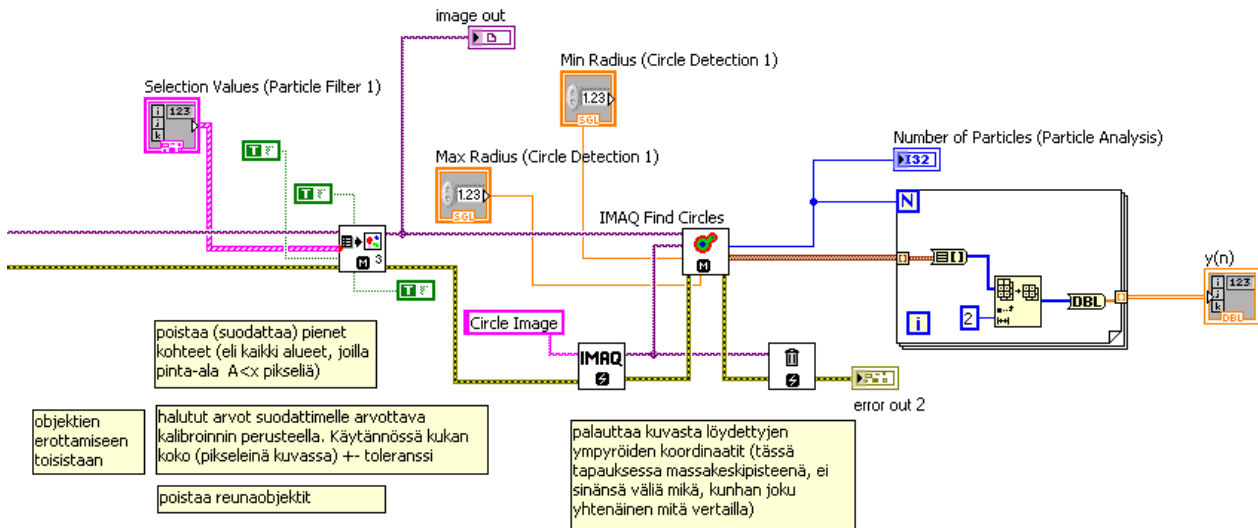


Figure 29: Pattern recognition for weed detection, second half. Filtering small, big and border objects, detecting circles of certain radius and returning their mass centre points

The coordinates of the found blobs are sent to a tracker which keeps track of the blobs and returns data that can be used for the weed handling device. The tracking algorithm is very simple, just looking for blobs on the right or left side of the image and checking that the coordinates of the objects in the frame have changed (Figure 30).

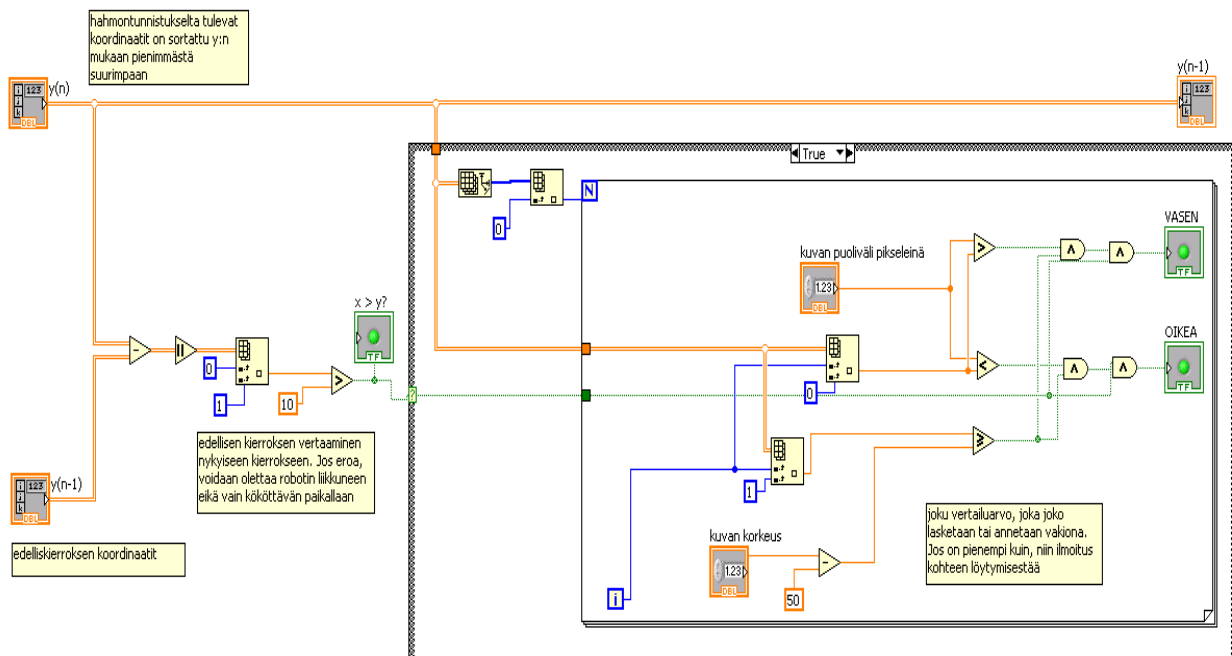


Figure 30: Tracking of the objects (weeds)

5. Programming techniques and communications

Several programming languages and development environments were used. Graphical programming in the form of Simulink Stateflow charts and LabVIEW was used. Bulks of code were generated automatically, but a lot of coding had to be done by hand as well. Program structure is described in Figure 31.

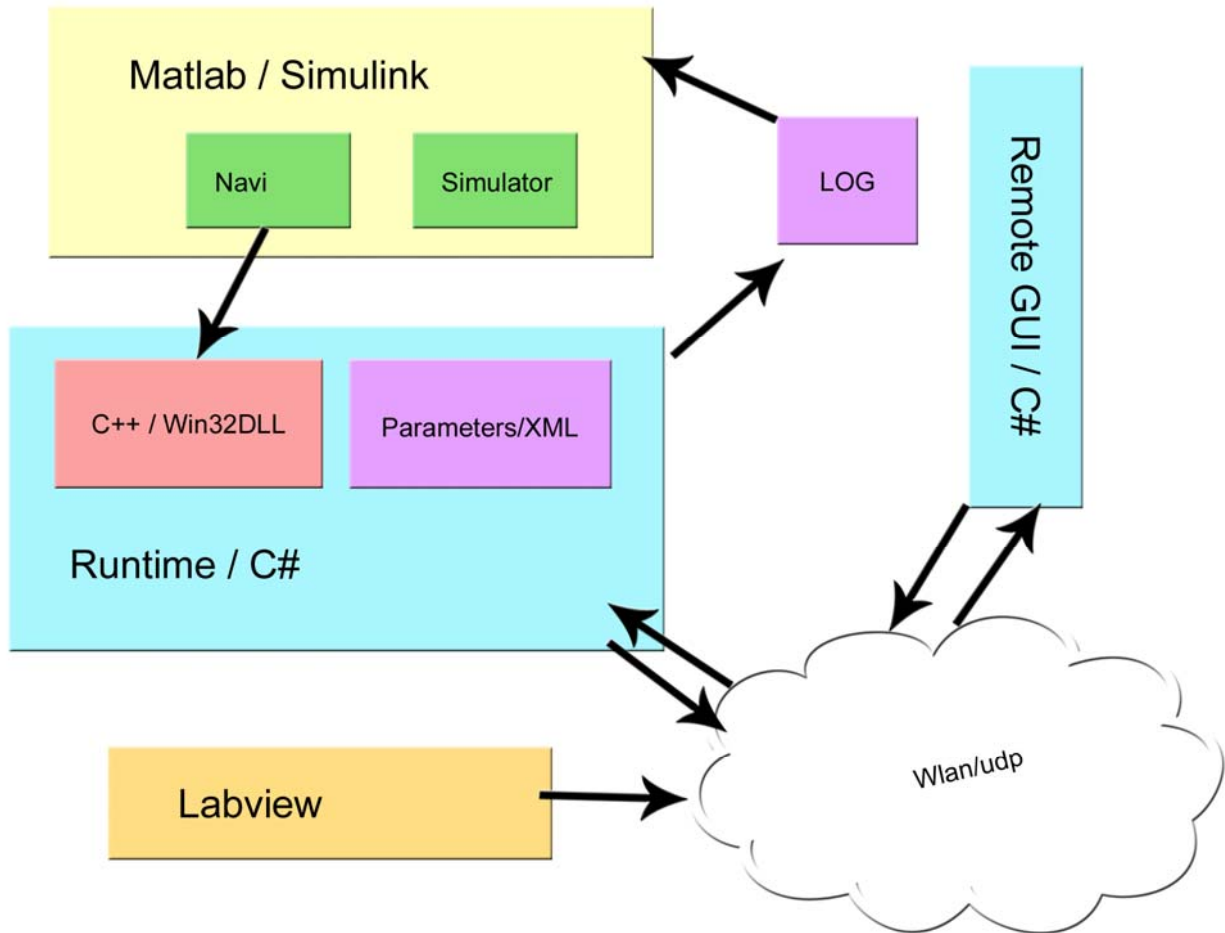


Figure 31: The program structure

5.1. Matlab & Simulink

The “brains” of the robot has been made using Simulink and also Matlab has been used for some functionality. The way to use Matlab and Simulink files in the robot is to generate them to C++ files. C++ files contain functions for running the program loop and definitions for inputs and outputs. Matlab version R2009b was used.

Matlab and Simulink have been also used for testing and simulation. The more about it is written in Chapter 6.2.

The Simulink model of the robots main program has been divided into functional groups, like row detection and row navigation. To make the model modular, Simulink libraries have been used for these functional groups. If someone makes a change to the library, the change will be updated to all models that use the same library. Of course the interface must be determined at first and it should not be changed. After that it is possible to develop different parts of the program at the same time by different people.

Libraries might contain other libraries or to be more precise, links to other libraries. We have used this possibility and whole navigation has been packed in one library block. The navigation and simulator blocks are shown in Figure 35.

Another important technique used is Simulink buses. Buses are basically signals that contain many signals. By using buses, it is possible to make interfaces to look simpler. Buses are also a good thing when compiling the model to C++. One bus is converted into one struct in C++ file, and all the signals in the bus are converted to fields into the struct.

Most of the used blocks are standard Simulink blocks, but also Embedded MATLAB Function blocks have been used. Some of them contain embedded code and some functions that have been made using Matlab m-files. It is not possible to use all Matlab functions because only part of them can be generated into C++.

The turning methods and some other features of the robot have been made by using Simulink Stateflow, which is a tool for drawing illustrative state machines. Stateflow is a good tool for describing a sequential action. An example about Stateflow is shown in Appendix 3.

5.2. LabVIEW and NI Vision

LabVIEW is a graphical programming language designed for measuring and automation applications. NI Vision is a platform of both software and hardware designed for machine vision applications. (National Instruments) Both of them were used to build the weed detection application for the field robot.

There are at least three approaches to developing machine vision applications with LabVIEW and NI Vision. The first is the basic approach: wire LabVIEW elements together and use the Vision library. With this approach, one has to know what he/she is doing. The second method is to use the Vision Builder application to build more complicated applications. For this project, the resulting LabVIEW code was far too complicated. The third method is to use Vision Assistant, a separate program which comes with the Vision package. For a novice developer it is probably the easiest approach to find most suitable algorithms and it was used for this project as well (Figure 28 and Figure 32).

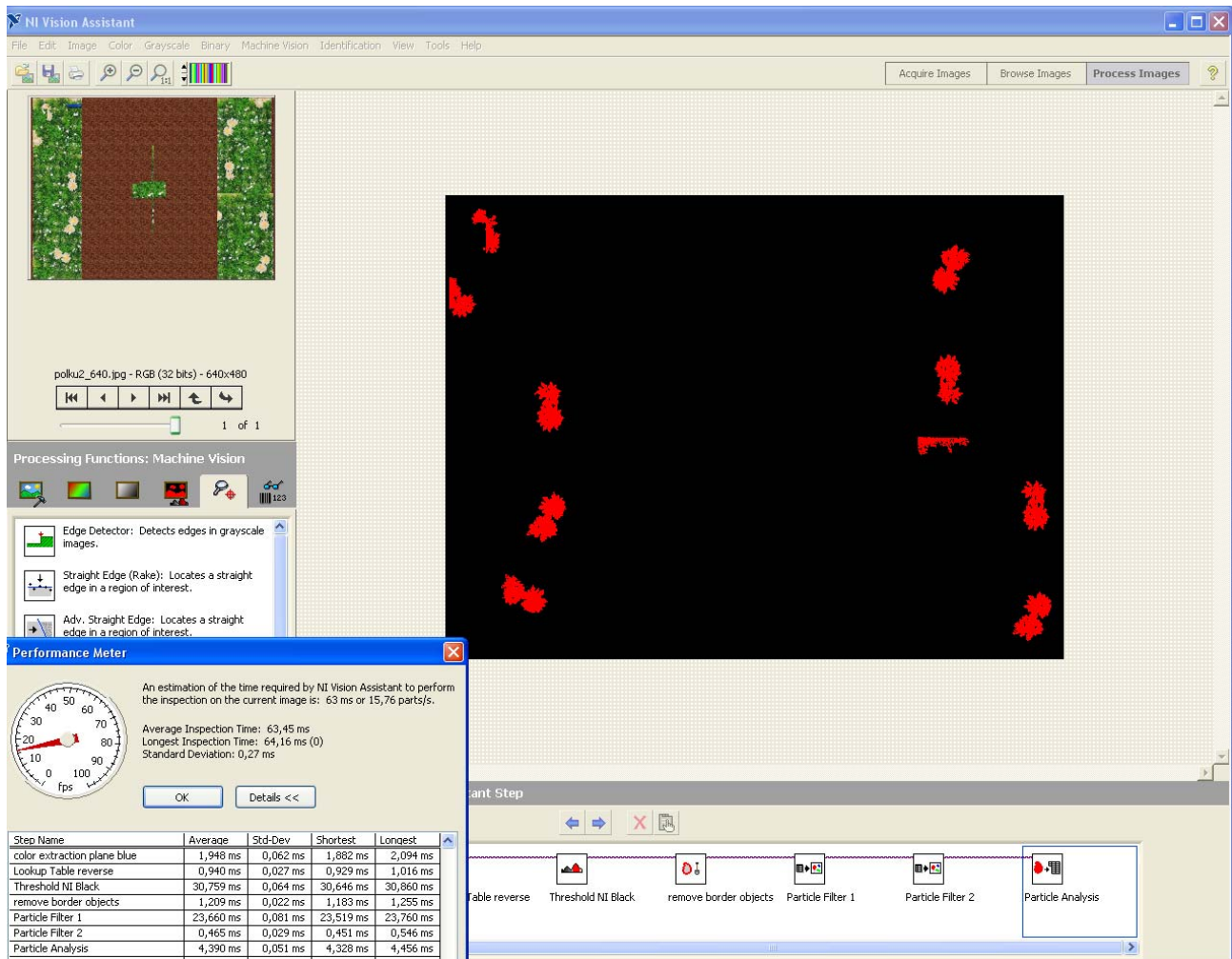


Figure 32: NI Vision Assistant with performance meter

Algorithms (scripts) were easy to develop and test for performance with the performance meter with Vision Assistant. Final script was easy to make into a LabVIEW virtual instrument ready to be used. The main program is a cut and paste of LabVIEW / NI Vision examples (Figure 33 and Figure 34).

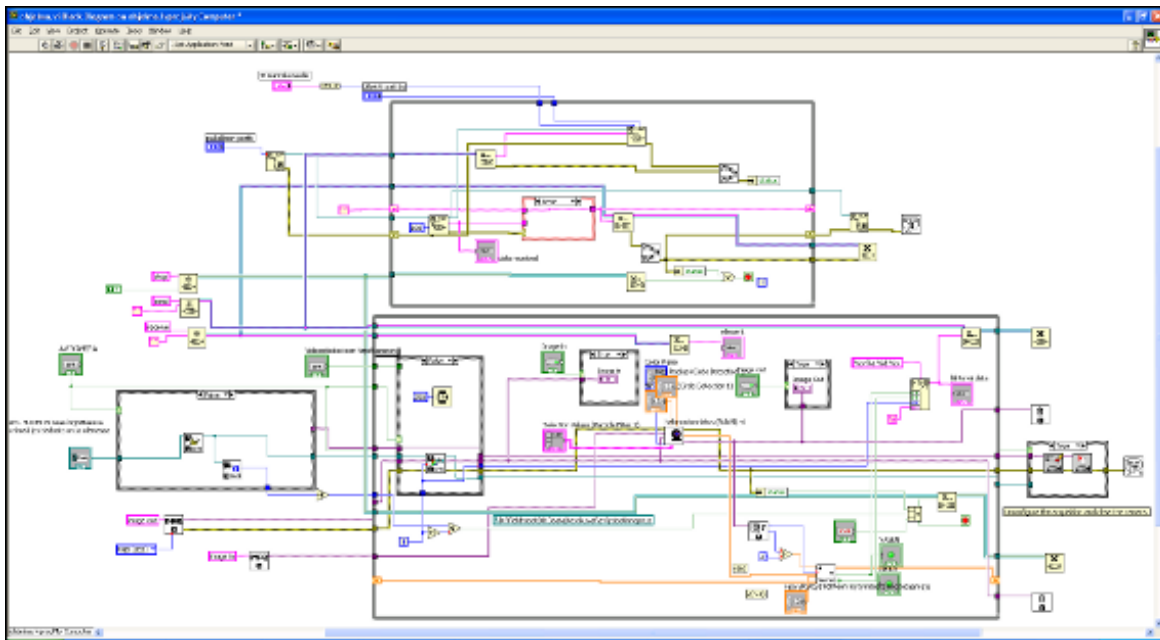


Figure 33: Example of program "code"

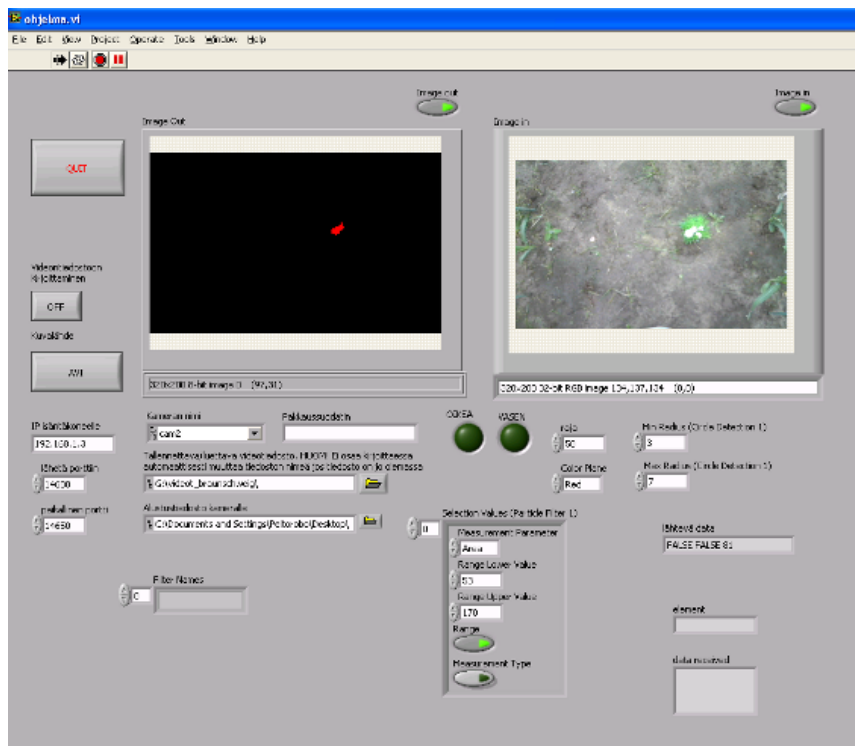


Figure 34: User interface

5.3. Visual Studio

Visual Studio 2005 and C# was used to build the main program, the remote user interface running on another laptop and communication with microcontrollers and remote user interface.

The main loop is based on Simulink RTW generated C++ code. The problem between different programming languages was solved by using P/Invoke. Basically it means that C++ code is generated as Windows DLL library. Functions in the library can be defined to C# code and called then as normal C# functions.

The idea behind this program structure is that when Simulink code is generated, it works directly without changes in the main code.

5.4. Telecommunications

Communication between the laser scanner, the main program, machine vision and the remote user interface is based on regular WLAN and LAN technologies. The robot contains a wireless router which is used as a platform for a standard IP network.

Communication between the laser and the main program is done with TCP/IP, as this is the way SICK scanner provides. All the other communication routes use simple UDP. Basically, every machine has their own predefined IP address and every communications message has its own port for UDP. When a program receives a message, the type of the message is identified by the port used.

Messages are basic C# structs which are serialized with Compact Formatter Plus before sending. Compact Formatter is used for changing structs to bit streams for UDP connection.

Communication with microcontrollers uses basic serial port. For this purpose a special "field robot protocol" was defined so that the developers of the C# code and C code have a common specification.

5.5. Parameters

We divided parameters into four different categories. Categories are explained in Table 1.

Table 1 - Parameter categories with their explains.

The prefix of category	Meaning
p0_	Parameter is constant and its value can be set only in Matlab e.g. The wheel base of the robot
p1_	Parameter might need some online tuning. It is possible to tune it manually in XML-file e.g. Warning limit for battery voltage
p2_	Parameter needs tuning. It is possible to tune it in the remote user interface. These parameters are common for every task in the competition e.g. Compass bias
p3_	Parameter needs tuning. It is possible to tune it in the remote user interface. These parameters are task specific e.g. Driving speed

Parameters in categories p1-p3 can be tuned and these parameters are stored in one common XML-file. This XML-file is stored in the robots onboard computer (eBox). Parameter values can be loaded from this file to the remote user interface whenever they are needed.

5.5.1. Parameter Slots

One nice feature used were parameter slots. In the program code one parameter “set” was written as a struct. In the eBox there was a list of these structs and structs were named as “Task 1”, “Task 2” et cetera. The main idea was that there was own slot for each task and all the parameters could be saved before hands. In the competition all you have to do is select the correct parameter slot instead of changing many parameters at once. This list was very easy to serialize to XML as described in the previous section.

Parameter slots proved to be successful in the competition and the system worked very well. However in the testing phase different parameter slots caused also some annoyance.

6. Testing

Our testing consisted of three parts: a simulator running in Matlab Simulink, artificial test field and of course individual testing for each part.

6.1. Individual/unit testing

Basically all the parts made had to be tested before use. Our testing method wasn't very precise and planned but before assembling bigger systems we tested individual subsystems. Axle modules for example, were tested thoroughly before they were attached to body.

The same methods were used with software. After making a sub routine, it is very important to test it because finding problems in bigger systems are somewhere between magic and wizardry.

6.2. Simulator

For testing our methods without the actual robot, a simulator library block was built in Matlab Simulink. Basically, it consists of basic kinetics which calculates robot's new position in our imaginary maize field and blocks which generate measurements for navigation algorithms used.

The simulator block is connected to the navigation block. This same block is used in the robot for its navigation. Of course it has been compiled to C++ code before using it in the robot. This structure is shown in Figure 35.

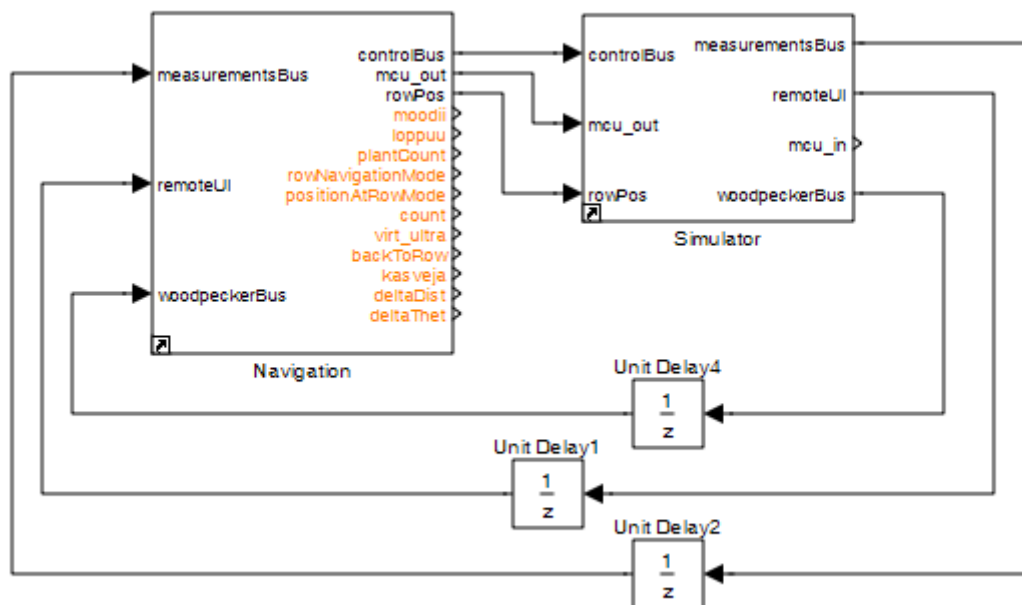


Figure 35: Connecting navigation library to simulator

At the beginning of simulation a field is created or loaded from a file. This field is plotted using basic Matlab *plot* command. After that robot is added to the same figure by using *fill* command. During the simulation Simulink model of the simulator (Figure 37) calculates a new position of the robot by using a kinematics model. The robot is moved to its new position by using figure handles and *set* command. It is computationally more efficient to “move” the robot than plot it again every time. The plotted user interface is shown in Figure 36.

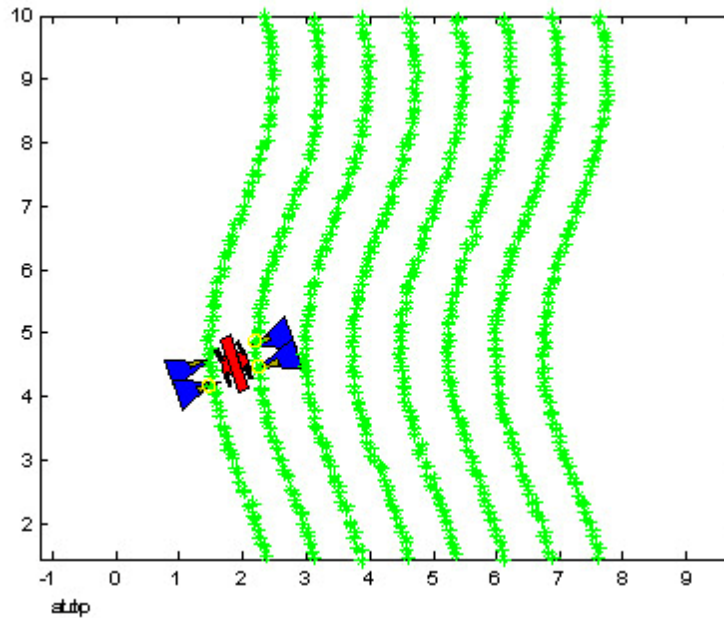


Figure 36: Simulator user interface

The simulator uses *find* and *inpolygon* functions to determine whether robot “sees” plants or not. The simulator block is not compiled to C++ so it is possible to use the wider range of Matlab functions than in the navigation block.

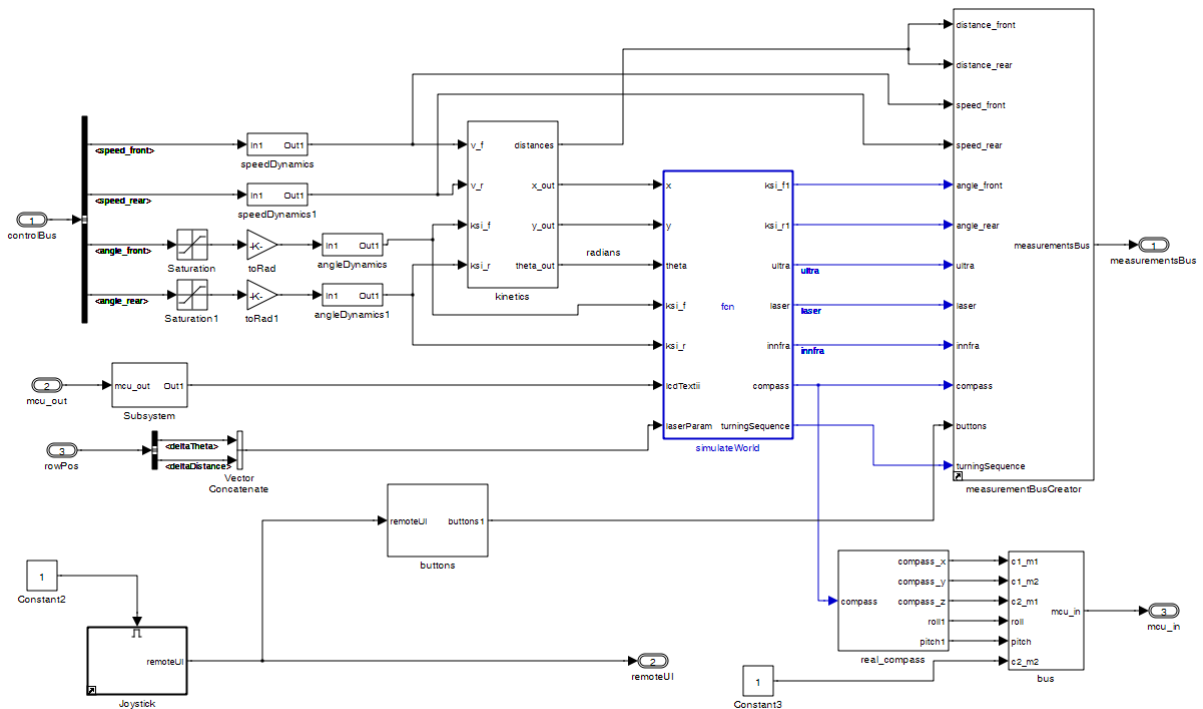


Figure 37: Simulink model of simulator block

6.3. Test Field

When the robot was constructed, we set up an artificial maize field at the back field of our university buildings (Figure 38). For the test field, a plastic construction fence was used and erected it with metal sticks. Construction was a very easy task and our hypothesis was that holes in the fence would generate enough noise for our measurements.

However, our artificial field was not that optimal. The plastic fence had some very nasty properties. Firstly if the robot hit the fence, it would suck the robot into it. A real maize row would have bent a bit. This was a very nasty feature because our sticks were made of steel so there were good chances to hit some iron with our €1000 laser scanner or ultrasonic sensors.



Figure 38: Test field in Finland

Second bad feature was vertical stripes and if the stripe was at the same level with sensors, there was not enough noisy signal.

Thirdly, the plastic fence easily makes waves horizontally and the amplitude of the waves can be 10 centimeters at the height of the robots sensors.

The biggest problem was the actual field in Germany (Figure 39). The spring was very cold and maizes were very tiny. In the test field, hardly any real maizes appeared and artificial sticks were put as a replacement. In the competition field, there were somewhat more real maizes but sticks were still used.

Anyway the tests were success in that sense that the robot performed pretty good when comparing with the other teams with less testing.



Figure 39: Competition field in Germany

7. Discussion

The robot participated in the Field Robot Event 2010 in June 2010 and was third in overall placing. The team had some major difficulties before the contest.

The schedule for building the robot stretched way over the set deadlines. A lot of work was done just weeks before the competition and even on the day before competition some development and debugging was done. It is difficult to say though if the robot had worked out better with more time or better time management.

When starting the project, it was difficult to tell what kind of things are needed and in which order they should be conducted. Many things were redone over and over again. Also some of the things were done to late because they were not recognized as 'bottlenecks'.

The robot's structure is quite complicated. For learning purposes this is good, but for better performance in actual competition, simplified structure would probably perform better.

We also used pneumatics for active suspension and for agricultural machinery. Active suspension with used construction was relatively weak but the main concept was a success. Probably it could be possible to actively maintain robot straightness in slope fields with this kind of structure, but for the compensation of quick movements our system is too slow. With agricultural machinery pneumatics worked fine.

Machine vision was made as simple as possible. Luckily, the weather was not too sunny so there was no need to adjust the camera for bright light. Unfortunately, the camera was still too low so that the field of vision was not wide enough and many of the weeds were missed. Because of the lack of the robust tracking algorithm, many of the initially detected weeds were missed when signalling the robot for weed control.

Testing in Finland was quite successful but basically almost everything broke down in Germany before the contest. Our compressor for pneumatics blew up twice and the third compressor was factory-made broken. The fourth generation was a combination of broken compressors. The main difficulty was a mystical electrification problem which made robots LEDs tingle like a Christmas tree and buttons to be pressed randomly. After changing one of the controllers and many hours our team found that one wiring to compass was somewhat loose which caused problems with I2C-bus. Two hours before the competition our machine vision computer broke down. Also we had some broken down ultrasonic sensors and WLAN-difficulties when both computers used wireless.

8. Conclusions

Most of the concepts tried in this experimental project worked as expected. However with some extra tuning and testing the robot could perform better and more reliably in the field.

Probably concepts of this kind are the future of the farming in more advanced countries. There are still many issues to be solved before robot's can perform their tasks autonomously on the fields.

The used development tools for the main program, Simulink, code generation, and Visual Studio were easy to use after some serious magic tricks in the beginning.

It was relatively easy to make the logic by using Simulink because one only needs to understand the block diagrams. It is also relatively simple to make a simulator where the navigation part can be tested. Very useful features in Simulink are: Libraries, Buses and Stateflow tool. Libraries make it possible to "write" modular and recoverable software. Busses are in the important role for making simple interfaces both in Simulink and in C#. Stateflow is very powerful tool for making software that runs in steps. Of course to be able to use these features in the robot, the code generation is needed. At the first time, some parameters needs to be set but after that the usage is very simple: just press one button.

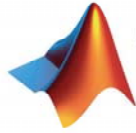
Visual Studio and C# language provide a combination that is quite easy to use. Especially tools for creating graphical user interfaces are very easy to use. Nevertheless there are also some complicated "tricks" to do before it is possible to use external C++ files created by Simulink. User for example has to add some path variables, manually modify a few files and give some compiler parameters. All these are things that user just has to know. After setting these things, using C++ functions still needs p/invoke method. In fact p/invoke method is very easy to use, all it needs are some special keywords. It can be say that these development tools form a very powerful combination and these tools should be first choice for a project like this. The only thing that was a disappointment was Windows CE (and Compact Formatter Plus). CE operating system should use only if other systems like Windows XP Embedded cannot fulfil strict real-time requirements.

Weed detection was simple but effective enough. With the test runs, it worked fine but for some reason, probably due to a communication lag between machine vision and robot, it failed to perform in a desirable way in the competition.

Acknowledgments

We would like to thank all our sponsors that made possible for us to do the project, and make a trip to the competition to take third prize in the overall competition. The sponsors were Valtra, Koneviesti, Suomen Kulttuurirahasto, SICK, FESTO, Laserle, HP InfoTech, Mathwork, Toradex, VTI. We would also like to thank our hosting universities: Aalto University and University of Helsinki.

Special thanks to Pekka Kumpula for designing the cover.



The MathWorks



HELSINGIN YLIOPISTO



FESTO

VALTRA



koneviesti

References

Hyötyniemi H.: *Multivariate Regression - Techniques and Tools*. Lesson 4. Page 66.
https://noppa.tkk.fi/noppa/kurssi/as-74.4191/materiaali/05._chapter_4.pdf

Backman, J., Hyyti, H., Kalmari, J., Kinnari, J., Hakala, A., Poutiainen, V., Tamminen, P., Väätäinen, H., Ok-
sanen, T., Kostamo, J., Tiusanen, J.: 4M - Mean Maize Maze Machine.
<http://autsys.tkk.fi/en/FieldRobot2008>.

Kemppainen, T., Koski, T., Hirvelä, J., Lillhannus, J., Turunen, T., Lehto, J., Koivisto, V., Niskanen, M., Ok-
sanen, T., Kostamo, J., Tamminen, P. Robot Brothers Easy Wheels and ReD in Field Robot Event 2009.
Proceedings of the 7th Field Robot Event 2009.

FireWire - Still the Performance King! <http://www.cwol.com/firewire/firewire-vs-usb.htm> - accessed June 30th
2010

Homepage of National Instruments. <http://www.ni.com/> accessed June 30th 2010

Appendix 1

```
function [thetas, R2, y2] = rlsmodulus(Fii, Y, lambda, theta0, P0, p3_maizeRowSpacing)
%#eml

% system dimensions
[n,m] = size(Fii);

% initial values
P = P0;
thetas = theta0;
R_2 = 0;

y2 = zeros(n,1);
f2 = zeros(n,2);

Index = 1;
maizes = p3_maizeRowSpacing;

for k = 1:n
    f = (Fii(k,:));
    y = Y(k);

    dX = inf;
    dY = inf;

    % plants that are not close to robot
    if(f(1,1) < -dX || f(1,1) > dX || y(1,1) < -dY || y(1,1) > dY)
        y2(Index,1) = 0;
        f2(Index,:) = [0 0];
        Index = Index +1;
    else
        f2(Index,:) = f;

        % divider (scalar value)
        S = f * P * f' + lambda;
        W = P * f' / S;
        P = P - W * S * W';

        ys = (mod(y+maizes/2 , maizes) - maizes/2);

        %for R2
        y2(Index,1) = ys;

        %scalar
        modulus = ys - (f * thetas);

        % evaluated parameters
        thetas = thetas + W * modulus;

        Index = Index +1;
    end
end

Sse = (y2 - f2*thetas)' * (y2 - f2*thetas); % eq 4.15
Sst = y2' * y2; % eq 4.16

if(SSt ~= 0)
    R_2 = abs(1 - Sse / SSt); % eq 4.14
else
    R_2 = 0;
end

% Check that 0<= R2 <= 1
if(isnan(R_2) || isinf(R_2) || R_2 > 1 || R_2 < 0)
    R_2 = 0;
end

R2 = R_2;
end
```

Appendix 2

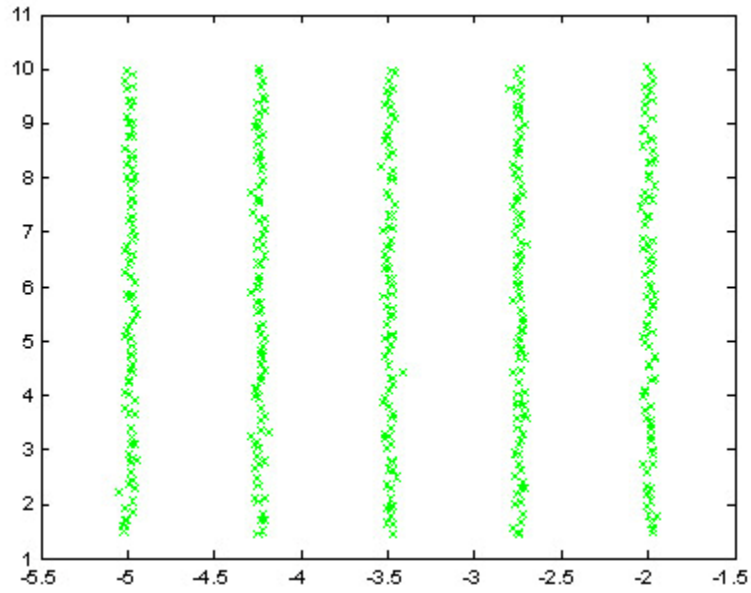


Figure 40: Simulated data from field

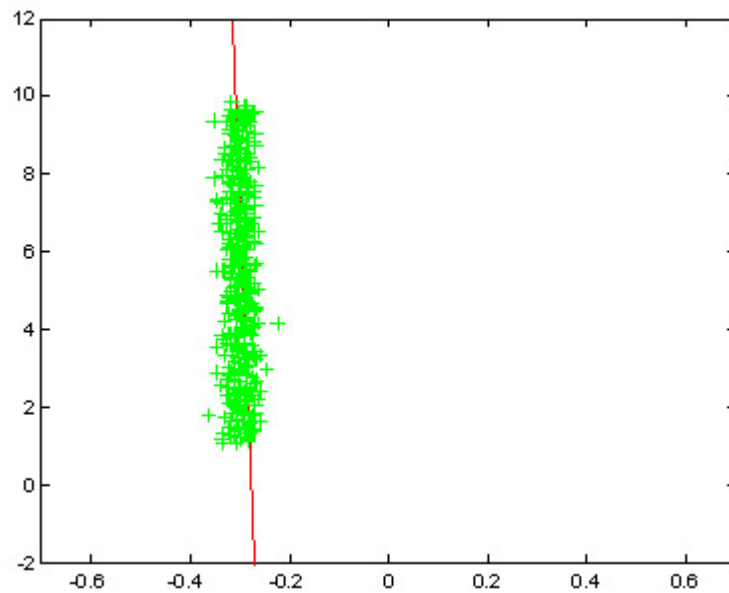


Figure 41: Optimal line fitting to data (modulus has packed data)

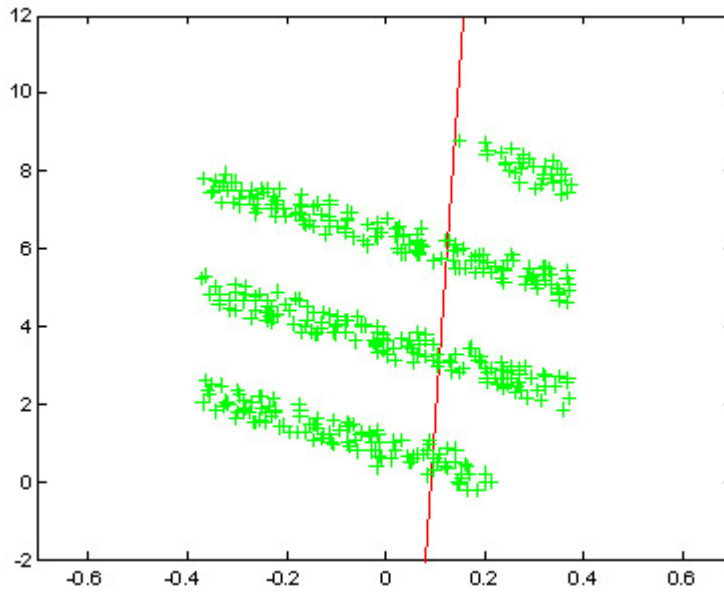


Figure 42: Angle between robot and rows is too big. The data folds and line fitting fails

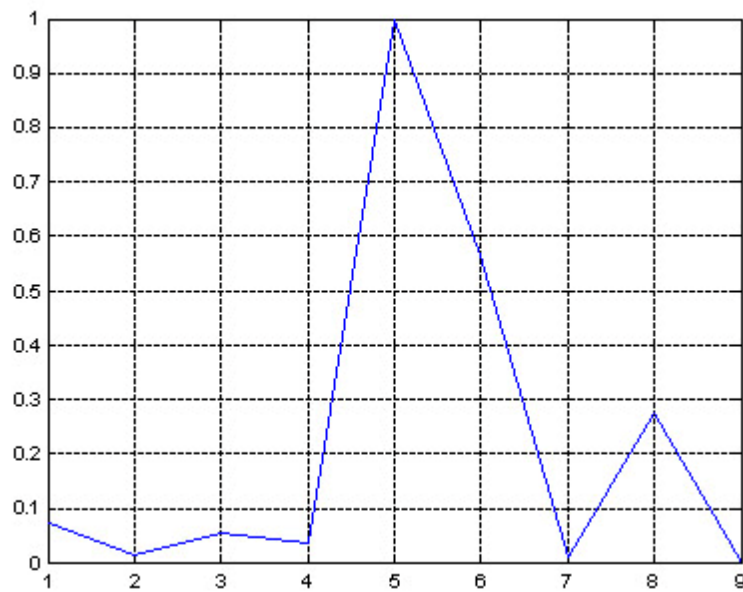


Figure 43: Correlation coefficient after different rotation angles. Number 5 represents situation in Figure 41 and 8 in Figure 42

